UNIVERSITY OF OSLO
Department of Informatics

Using Ultra Wide Band Impulse Radar in Simultaneous Localization and Mapping

Master thesis

Jonas Bøhn Løchen

May 1, 2013

Abstract

Ultra Wide Band Impulse Radar (UWBIR) is a technology that uses electromagnetic waves for ranging. This has been true for radar systems throughout the 20th century, although their applications in mobile robotics have been limited due to the physical size of the systems. The much more recent UWBIR technology enables radar systems to overcome this limitation, and therefore has many potential application areas.

This thesis investigates if a mobile robot can map and safely navigate in an unknown indoor environment by using Simultaneous Localization and Mapping (SLAM) relying on a distance sensor based on the UWBIR technology for ranging. Traditionally, LIDAR and other optical sensors, as well as sonar, have been the preferred sensors for mobile robots. UWBIR technology has only recently been introduced in stand-alone ranging-operations, and few attempts have previously been made at utilizing such sensors in SLAM. Compared with LIDAR, an UWBIR system has the advantage that it contains multiple echoes in its range data, as well as having the ability to operate in occluded environments. Its drawbacks are that it can suffer in resolution and computational effort.

For evaluating how well UWBIR performs in SLAM, raw radar data is sampled by a mobile robot equipped with an UWBIR distance sensor while moving through real indoor scenes. Using a combination of several signal processing algorithms like clutter maps and bandpass filtering, this thesis shows how features can be *extracted* from the raw data and combined with positional data to generate maps of the robot's environment, as well as produce through-wall images. Further, machine learning algorithms like neural networks and support vector machines are shown to be able to *classify* the material of an object by its pulse signature in the UWBIR data with great levels of accuracy. Finally, an implementation of a mapping-algorithm using the extracted and classified features is proposed.

The results gained in this thesis leads to the conclusion that UWBIR can not compete with more traditional computer vision systems like LIDAR as a stand-alone sensor technology due to lack in resolution. However, they motivate putting further efforts into combining UWBIR and SLAM through e.g. imaging radar systems or sensor fusion with other sensor technologies like LIDAR. In such settings, UWBIR can be used for sensing in occluded environments and classifying materials and objects, as well as through-wall imaging.

Contents

Preface

This thesis is submitted as part of the degree Master of Science in Robotics and Intelligent Systems at the University of Oslo.

Despite several iterations with my excellent supervisors - Professor Jim Tørresen at the University of Oslo and M.Sc. Elias Bakken, CTO at Intelligent Agent AS - in narrowing down the scope and focus of the thesis, the project turned out to be quite a workload. Several months were spent finding, purchasing, assembling and modifying parts for the mobile robot used for the experiments performed in the thesis. Parallel to this was writing firmware and software for the robot. When it was all assembled came another several-months period of testing and modifying the hardware and programs, until the platform finally was ready for actual data recording (never would I have thought at the outset that I would have to design and order a circuit board!) Without the invaluable assistance from Elias Bakken and CEO at Intelligent Agent AS Øyvind Dahl during these periods, no experiments would likely ever have been performed, and simulated data would have had to suffice.

Ultimately, only about half of the desired time to spend on signal processing and developing classification algorithms remained. Despite these delays, I am pleased with the many results and experiences gained when attempting to combine UWBIR sensor data and SLAM. I would like to thank Professor Jim Tørresen for the large amount of feedback and suggestions for improving both results and master thesis, which allowed the project to land safely.

Thank you also to the good people at New Generation Communication AS for cheering me on and putting up with me redecorating their office time and again in order to perform experiments. Last but definitely not least, a huge thank you to my girlfriend Jeanette and my family for continuously inspiring me and putting up with such a preoccupied mind for 18 months.

Part I

Introduction

1 Robot navigation

For an unmanned vehicle to be fully autonomous, its ability to navigate safely and dependently within its operating environment is crucial. This is no trivial task in itself, and when no a priori knowledge of the environment exists, the difficulty of the task is further increased. A family of algorithms for solving this problem is called *Simultaneous Localization and Mapping* (SLAM). Originally developed by Hugh Durrant-Whyte and others [5][6], it is an important part of many robotic applications today. It can be described as the process of combining input from a variety of sensors to answer the two questions: 'What does the world look like?' and 'Where am I in it?'

The robot should be able to answer these two questions with a high level of precision at all times. Physical aspects make this hard to achieve; inaccurate or non-existing data, latency and noise in data from one sensor affects all other sensors, and several other factors. Therefore, it is always interesting to evaluate how well sensors utilizing new technologies are suited for operating in a SLAM-setting.

1.1 Motivation

Mobile, self-sufficient robots have been on the rise throughout this millennium, a trend which shows few signs of slowing down [14]. In the commercial market, lawn mowers, vacuum cleaners and micro helicopters amongst others are all well established products and are still growing businesses, while the automobile industry introduces more and more autonomous elements to their series. In military markets, reconnaissance drones have been referred to as 'game-changing' [24] in the field.

All of them rely on extracting information about their environment, and most likely utilize some form of SLAM-algorithm. A robot implementing these algorithms is capable of both mapping its surroundings as well as estimating its own position in this map, given accurate sensor data. These operations go hand in hand, and robustness of all sensor measurements is therefore essential.

In order to be able to perform SLAM, the mobile robot must rely on inputs from a number of sensors, both proprioceptive (sensing the robot's state) and exteroceptive (sensing the robot's environment). Both categories have a multitude of different sensors and technologies, but most relevant for this thesis is to introduce the different exteroceptive in particular.

By far the most common exteroceptive sensor is *LIght Detection And Ranging*,

or LIDAR [19]. This technology uses laser emissions to determine what the agent's surroundings looks like. It is a very accurate ranging, given that the operating environment of the agent is non-occluded. In case of fog or smoke the LIDAR-approach is useless. Other optical-based technologies include cameras and stereo vision [10], which means capturing the scene with two cameras simultaneously and extracting the geometry by comparing these two. This approach is also impeded by occlusion. A common technology for being able to extract information about the environment no matter of occlusion is *SOund Navigation And Ranging*, or sonar. This is a non-optical approach where sound waves are emitted from the sensor and the echo gives the distance to the obstacles around the robot. This is a cumbersome way of ranging, due to poor angular resolution [25]. An alternative to sonar is using radar-based sensors, which similarly emits *electromagnetic* (EM) waves and uses the time delay between the wave was emitted and a reflection is received to calculate the distance. Traditionally, radar systems have depended on large apertures and continous EM waves for ranging. In 2002, however, the radar frequency (RF) band 3.1-10.6GHz was opened for commercial use. This made it possible to use a type of radar called *Ultra Wide Band* (UWB) *impulse radar* (UWBIR). This is, in short, a nanoscale radar capable of very high resolutions by emitting short electromagnetic impulses.

1.2 Outline of thesis

This thesis aims to propose an implementation of a SLAM algorithm using an UWBIR distance sensor, and analyze its performance with regards to accuracy and robustness. The various aspects of the thesis are described in dedicated chapters, of which a brief outline is presented below.

Chapter 1: ***Robot navigation*** is this chapter, which has presented a brief overview of the thesis and the motivations behind it.

Chapter 2: ***Simultaneous Localization and Mapping*** defines and explains the SLAM-problem and presents both historical and more modern approaches for solving it.

Chapter 3: ***Ultra Wide Band Impulse Radar*** introduces the UWBIR technology. Raw radar data from an UWBIR system and ways of processing it is described.

Chapter 4: ***Data association with UWBIR*** continues the focus of the previous chapter; here, classification of objects present in the processed UWBIR data is presented.

Chapter 5: ***Setup of experiments*** outlines the practical aspects of the thesis, i.e. which experiments will be performed and with what hardware and software setups.

Chapter 6: ***SLAM with distances*** shows the results from using pure distance measurements from the UWBIR system used in this thesis for performing SLAM.

Chapter 7: ***SLAM with raw data: Feature extraction*** evaluates how well optimized signal processing algorithms succeeds in extracting features from raw UWBIR radar data, and discusses whether these features combined with the positional data from the robot used in the experiments alone are enough for performing SLAM.

Chapter 8: ***SLAM with raw data: Feature classification*** presents the results from attempting to classify UWBIR data with respect to materials present in the scenes in which the robot operated, and investigates how this can be used to enhance SLAM with an UWBIR system.

Chapter 9: ***Conclusion*** sums up all the results and discussions presented in the thesis, and evaluates whether UWBIR technology can be used for achieving autonomous robot navigation.

Appendix A: ***Matlab scripts*** shows the signal processing algorithms and other Matlab scripts used in this thesis; this is only a small part of the code developed, but it is the only 'user-friendly' enough to read, as much of the other code is patched into existing source codes with thousands of lines.

Appendix B: ***SLAMbot schematic*** shows the full schematic of the SLAMbot connection board designed in this thesis.

Part II

Background

2 Simultaneous Localization and Mapping

In this chapter, SLAM will be defined and two separate ways of performing it will be presented: extended Kalman filtering and particle filtering. The theory is, unless otherwise stated, described in greater detail in [5] and [6]. This thesis only covers 2D-SLAM.

2.1 The SLAM problem

SLAM is the process of combining sensor data from a mobile robot to make a map of the robot's surroundings as well as determine its location within the environment. The robot is typically equipped with one or more proprioceptive sensor(s), e.g. wheel rotation encoders or GPS. These report how the robot moves, i.e. yields *positional* data. Additionally, the robot is equipped with exteroceptive sensor(s) for measuring distances or otherwise sample the robot's surroundings. These include laser range finders, cameras and sonar, and their outputs will from now on be referred to as *ranging* data. The positional and ranging data are combined to perform the mapping, and it is this mapping that is referred to as the SLAM *problem*. The solution to the problem has been referred to as a "holy grail for the mobile robotics community as it would provide the means to make a robot truly autonomous" [5]. Such solutions exist today, and will be examined later in this chapter.

Figure 2.1 shows a flowchart of a SLAM algorithm. From the collected (and possibly processed) sensor outputs, positional data is extracted and used to determine the current pose and position of the robot relative to the previous. This is made possible by the *motion model*. This is a model of the robot itself and how it reacts to the movement described by the sampled data, along with the history of previous positions. From the exteroceptive data, *features* or *landmarks* are extracted. These are combined with the features extracted from previous measurements, which forms a map through the *observation model*. Lastly, the updated models are filtered and used to correct each other.

These steps will be examined in detail throughout this chapter, and the following section will further define the motion and observation models.

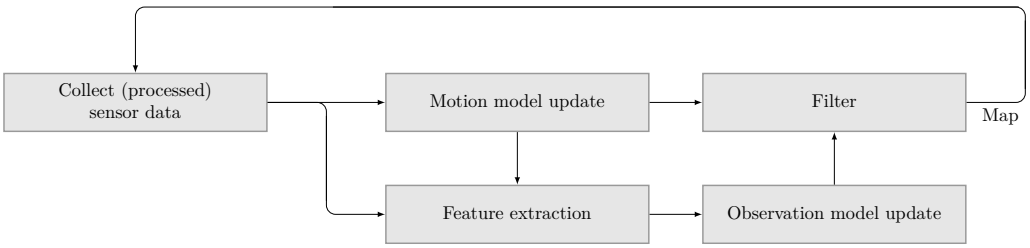


Figure 2.1: SLAM flowchart.

2.1.1 The probabilistic SLAM problem

The SLAM problem is in the general case defined in a probabilistic form. For a mobile robot observing an unknown environment as illustrated in Figure 2.2, the sensed world can be quantized to sets of observations ('Collect sensor data' in Figure 2.1) from which landmarks can be extracted ('Feature extraction' and 'Filter' in Figure 2.1). Additionally, the robot's pose and position (*state*) in the environment is quantized ('Collect sensor data' in Figure 2.1). SLAM is performed by combining these discrete quantities, where the set of landmarks becomes the map of the environment and the set of robot states becomes the track the robot has followed in it. Figure 2.3 illustrates the interaction between the motion model and observation model as introduced in the previous section. The exteroceptive measurements can only form a good map if the pose of the robot is correct, but the motion model yielding the robot pose depends heavily on relating the current exteroceptive measurements to the existing map. SLAM is therefore the process of combining all sensor measurements in a way that maximizes the likelihood of the generated estimates being correct. This is achieved through defining the models as the probability pools

$$P(\underline{x}_n|\underline{x}_{n-1}, \underline{u}_n), \tag{2.1}$$

and

$$P(\underline{z}_{n,i}|\underline{x}_n, M), \tag{2.2}$$

where $\underline{z}_{n,i} \in Z$ are the observations, M is the set of landmarks and $\underline{x}_n \in X$ are the robot positions, where $\underline{x}_{n+1}$ represents the next state of the robot when a force $\underline{u}_n$ is applied. These distributions the probabilistic motion model and observation model of the robot respectively, and are continuously updated during the execution of the algorithm. There are two common methods of extracting these distributions, namely *extended Kalman filtering* (EKF) and *particle filtering*. Both will be used in this thesis, and are described in detail later in this chapter.

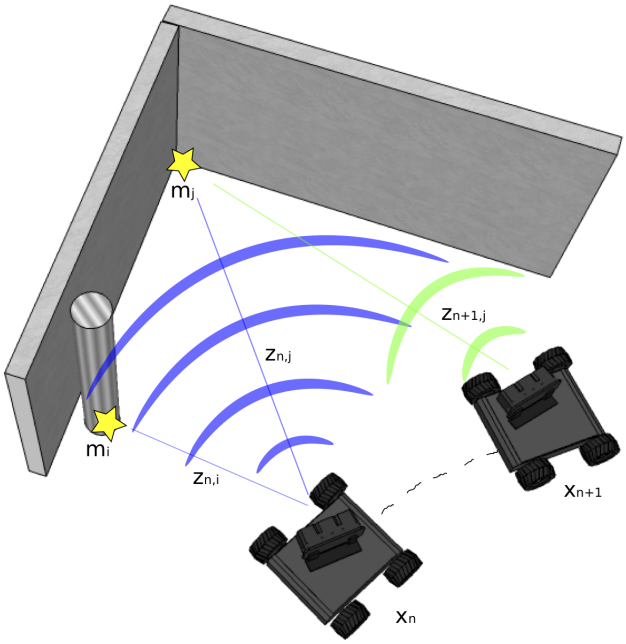


Figure 2.2: The SLAM problem.

2.2 Main challenges in SLAM

Even though the SLAM problem is solved, there still exist many challenges and potential improvements.

2.2.1 Data association

A big problem in SLAM is the mapping between observation and landmark. The essence of this problem is to identify landmarks in different samples, and not generate more landmarks than are actually present.

2.2.2 Computational complexity

Performing SLAM is an operation that should be performed real-time. This greatly limits the scope of the algorithms. The nature of the SLAM-problem is, regardless of which approach is used, to discretize the environment of the robot by making measurements. For each measurement or sample made, combine it with previous measurements to update the representation of the environment. These inter-sample calculations limit the sampling rate.

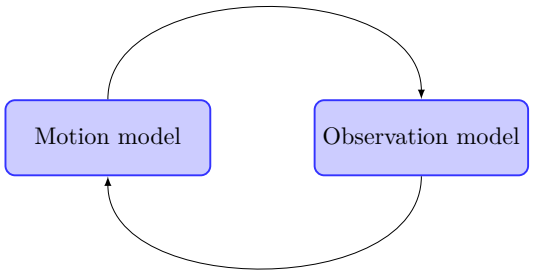


Figure 2.3: The motion model and observation model depend on each other.

2.3 Extended Kalman Filtering

The Kalman filter is one of the most used algorithms in tracking and navigation [13]. A pure Kalman filter is only suited for linear problems, while the extended Kalman filter removes this limitation by linearizing the input; this will be further discussed in the next section.

The objective of the EKF is to use a series of measurements to produce estimates of unknown variables for describing a system. This is done by combining a priori information about the system (a motion model) with the a posteriori information about the environment (an observation model). In a perfect setting, the system model would tell us exactly how the robot move at all times, and the observation model would contain exact information about what the surroundings of the robot looks like. In the real world this will not be the case, as all observations and measurements are noisy, hence corrupting both the system and observation model. This is the reason we call SLAM a probabilistic problem, as we can never be certain that we produce an exact model of the environment, we can only strive to maximize the probability of its validity. For white noise (e.g. zero-mean, Gaussian distributed noise caused by the myriads of small reflectors in any sampled scene), the Kalman filter is an optimal estimator. This means that the output from the algorithm will be a minimum-variance estimate of the variables described by the *Gaussian distributed* motion- and observation models. This is exactly what we seek to accomplish in SLAM, where we know that the majority of the noise in the observations is statistically white [12][4].

Figure 2.4 illustrates the basic mechanics of an EKF. The physical process is controlled through the applied force $\underline{u}_n$, and affected by the noise $\underline{v}$, resulting in a state $\underline{x}_n$. This state is sampled by a sensor with a noise described by $\underline{w}$, resulting in an observation $\underline{z}_n$. The EKF is made up of models of the physical process and sensor, which try to estimate what the next states and therefore observations will be ($\tilde{\underline{x}}_n$ and $\tilde{\underline{z}}_n$). When the new samples are available, the difference between the estimated and the real observation is amplified by the Kalman-gain KG. This gain is adaptable, and its value depends on the accuracy of the filter thus far, generally described by the mean

and covariance matrices of the observations. Lastly, based on the difference between the estimated and observed values, error terms for the process model is generated and fed back to the closed loop. These are used to update the process and sensor models. This way, the filter is iteratively updated and improved by gaining accurate models for the noise in the real-world process and sensor. It is therefore able to produce minimum-variance estimates of the actual process. The actual output from the filter is the estimated state $\underline{\tilde{x}}_n$ for each iteration.

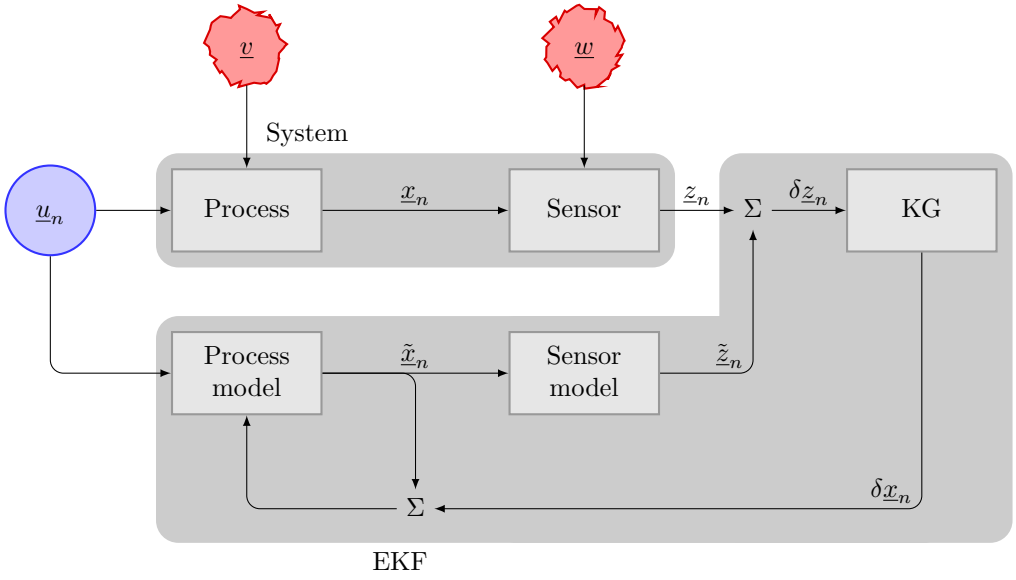


Figure 2.4: Extended Kalman filter flowchart.

2.3.1 EKF-SLAM

In a SLAM-setting, we rewrite the motion model and observation model as shown in equations 2.1 and 2.2 to

$$\underline{x}_n = f(\underline{x}_{n-1}, \underline{u}_n) + \underline{w}_n, \tag{2.3}$$

and

$$\underline{z}_{n,i} = h(\underline{x}_n, M) + \underline{v}_n, \tag{2.4}$$

respectively, where $f(\cdot)$ and $h(\cdot)$ describes the robot's kinematics and the geometry of the observation respectively. w_n and v_n are additive, zero-mean Gaussian noise. Where the models previously were given only as probability distributions, they are now because of the properties of the EKF simply functions of measurable quantities (i.e. outputs from the robot's sensors). Hence, an EKF is implemented by linearizing the differentiable $f(\cdot)$ and $h(\cdot)$ functions, so that we are able to produce a minimum-variance estimate for both the environment (M) and the robot's location in it (X).

2.3.2 Challenges in EKF-SLAM

One of the greatest difficulties with this approach to performing SLAM is data association. Only the slightest misclassification of a landmark can potentially make all later mappings between observation and landmark erroneous, because of the fact that this entire approach is based on the covariances between measurements. One 'faulty' sample $\underline{x}_n$ or $\underline{z}_n$ has the ability to distort the robot's entire model of the environment, and can not be removed once added to the history of samples (Z, X). Even in the less extreme case, where observations only differ slightly from what the world actually looks like, this becomes a problem when the robot returns to a previously observed area after a (long) trek in the rest of the environment. Now, the model can be quite accurate but still be shifted compared to the real world, making the robot unable to identify the area as previously seen. This is known as the *loop-closure problem*, and it is best avoided by putting extra effort into avoiding misclassifications. In [6], methods for doing this are described, one of which is to not only look at the observed geometry in each sample $(h(\cdot))$, but also take into account other qualities of the sample (e.g. color/texture for stereo vision or pulse signature for sensors based on sound or electromagnetic pulses).

2.4 Particle filtering

A more recent approach to SLAM is using Monte Carlo methods, also known as particle filtering. The methodology uses EKFs, although in a different way than EKF-SLAM, as will be reviewed in the next section. The main difference between EKF-SLAM and particle filtering is that instead of estimating poses and positions based on the full history of the algorithm, particle filtering estimate several parallel trajectories, known as *particles*. These particles have separate models, i.e. probability distributions, based on the current measurements with random noise and the previous 'absolute' pose. Thus, several optional trajectories are estimated simultaneously and can be evaluated separately. Many such particles form a probability model of the robot states not bound by the Gaussian assumption of the EKF-approach.

2.4.1 Probabilistic particle filtering

In EKF-SLAM we describe the system as one continuous EKF for robot states and observations; every update $\underline{x}_n$ is estimated based on any current measurements and the current mean and covariance matrix of the state set X as well as landmarks M. These covariances are most often quite large, because both pose- and landmark-observations are only estimates themselves.

Particle filtering uses another methodology. The process is illustrated in Figure 2.5, and explained throughout this section. Instead of using computational effort on finding the minimum-variance estimate of the robot's pose based on the entire set of previous robot positions (or *trajectory*) and landmarks, a number of estimates are kept simultaneously. Using the terminology used above, while EKF-SLAM only estimate one

trajectory consisting of many estimated states,

$$\underline{x}_n \in X, \tag{2.5}$$

in particle filtering several trajectories are estimated while the robot operates,

$$\underline{x}_n^{(i)} \in X^{(i)}, \tag{2.6}$$

where i is one of N particles. The main difference from EKF-SLAM is that while the EKF-approach estimates one new pose every update, $\underline{x}_n$, and the uncertainty of this estimate is added to- and changes the total covariances of the system models, particle filtering uses each new update $\underline{x}_n^{(i)}$ as an 'absolute' pose. The new pose is drawn from a suitable probability distribution based on the measurement and the current trajectory $X^{(i)}$ so far, which varies from particle to particle. Thus, there is no uncertainty being estimated about the pose and added to neither motion model nor observation model, and the set of poses become conditionally independent. This implies that the set of landmarks each particle keeps also can be evaluated independently, while the EKF must for each new observation update all landmarks and covariance matrix.

Each particle keeps its own map with landmarks,

$$\underline{m}_n^{(i)} \in M^{(i)}, \tag{2.7}$$

where the observations are filtered with e.g. EKF. Further, each particle is weighted based on their individual motion models and observation models. At a certain interval, the particles are resampled by choosing only the most 'fit' particles, so as not to keep the particles that potentially have started to drift. This is illustrated in Figure 2.6, where the dotted line shows a 'correct' trajectory, and the other lines are estimated trajectories, i.e. particles, normalized to the correct one. Halfway through the sampling, only those particles with variations below a certain threshold are kept, resulting in a steadier convergence. Finally, when a loop is closed or the algorithm meets some other criteria, we have several possible trajectories to choose from to represent the final SLAM-products M and X.

2.5 Example maps

As previously stated, SLAM is a well established family of algorithms, and there are plenty of 'plug-and-play' implementations to be found. For ease-of-use, two of these were chosen to be used in this thesis, instead of developing completely new algorithms. These two are the CAS-toolbox [2] and DP-SLAM [9], which use EKF-SLAM and particle filtering, respectively. They will be examined in section 5.5.

Included with the CAS toolbox are example data sets, whose data is collected using a robot equipped with rotary encoders and a laser range finder (shown in Figure 2.7), similar to the robot used in this thesis. From one of these data sets, the CAS toolbox is able to generate the maps shown in Figure 2.8a/b, where (a) visualizes the filtered

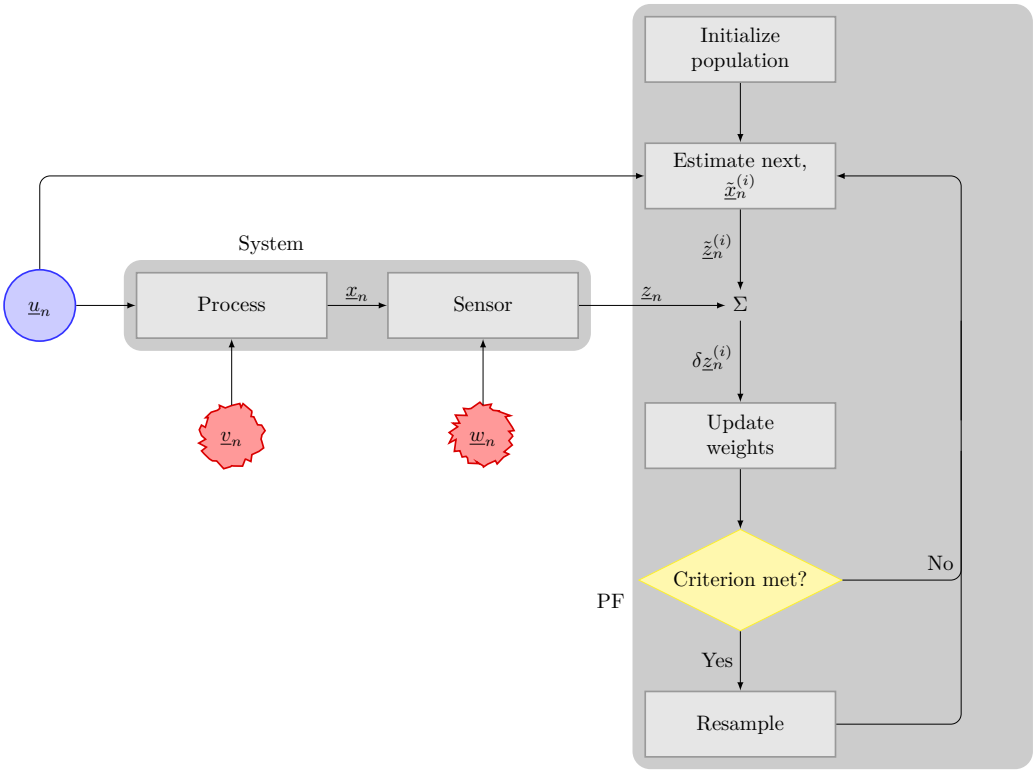


Figure 2.5: Particle filter flowchart.

raw data (visualization of the surroundings) and (b) shows the landmarks and lines the algorithm extracted (navigational map).

After extracting the complete odometry data from the rotary encoders, the same data can be used directly in the DP-SLAM algorithm. The resulting map is shown in Figure 2.8c. It displays an equally good performance, without extracting any landmarks.

These examples illustrate how well the two algorithms perform. In later sections we will examine these algorithms further and examine how well they perform with ranging data from UWBIR systems.

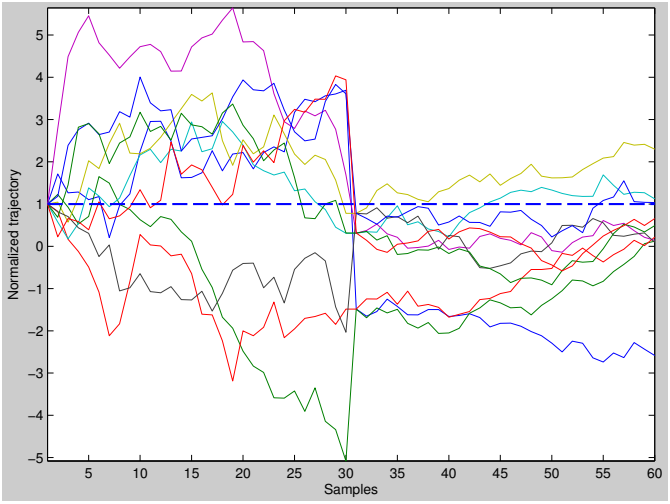
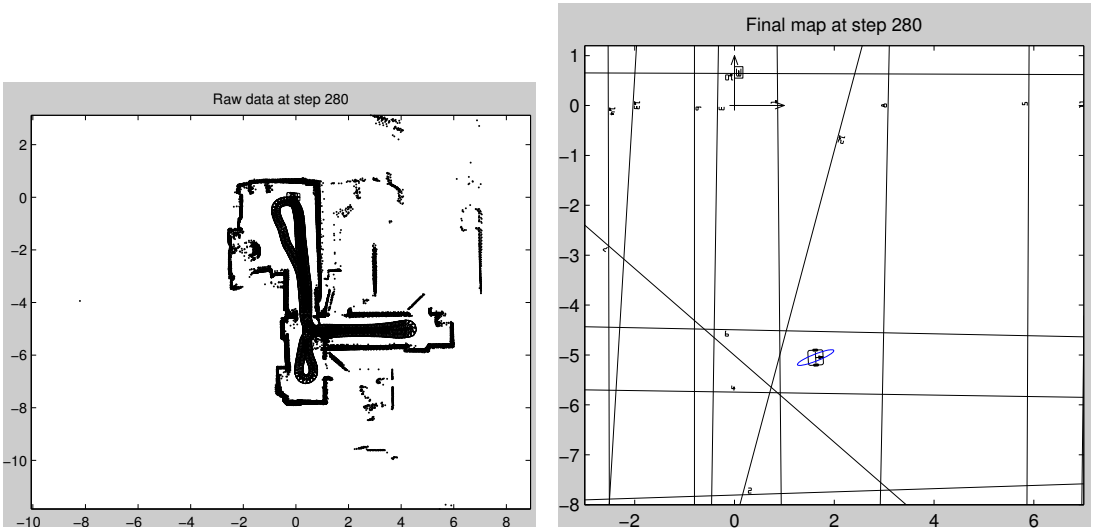
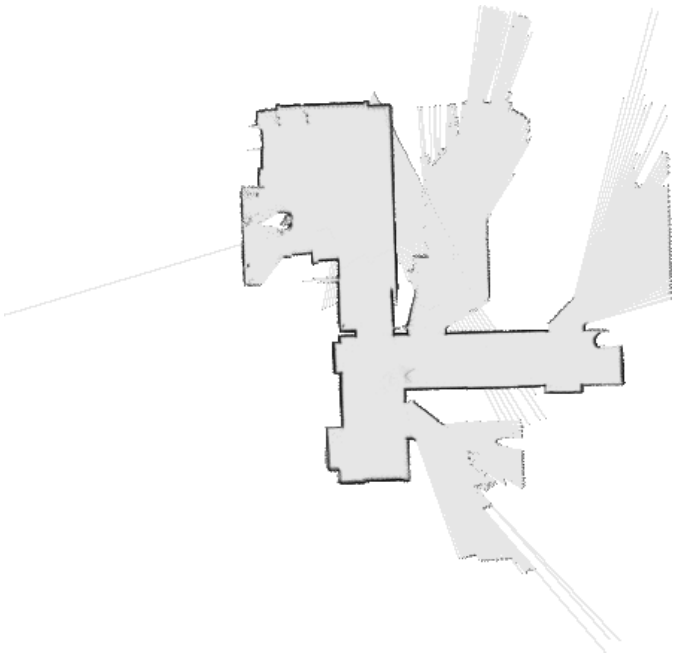


Figure 2.6: Trajectory-estimating particles.

Figure 2.7: The SmartRob robot used for collecting data in [2].



(a) (b)



(c)

Figure 2.8: Example maps generated by the SLAM-algorithms: (a) CAS toolbox visualization of surroundings, (b) CAS extracted landmarks/lines, (c) DP-SLAM visualization of surroundings.

3 Ultra Wide Band Impulse Radar

This chapter introduces the UWBIR technology, and discusses how information is extracted from the raw data produced by such systems. A more thorough introduction and historical overview can be found in [4] and [11].

3.1 Ranging with UWBIR

One of the main advantages with using sensors that uses EM-pulses for sampling is that fog and other occlusions do not affect the returned signal, as compared to for example LIDAR sensors. Figure 3.1 shows a top view of a room in which a robot equipped with both an UWBIR-sensor and a LIDAR-sensor is placed. The returned distances from both sensors are plotted; in (a) the room is fog-free, while (b) shows the returns in a fog-filled room. The UWBIR-returns are similar in both settings, while the LIDAR returns are made much worse by the foggy environment. The LIDAR returns from the fog-free room shows a better depiction of the surroundings, while the reported distances from the UWB radar are somewhat curved. This can be caused by several things; most likely it is a result of that while the LIDAR samples by transmitting a multitude of strobes at slightly different angles, the radar sensor works by transmitting a pulse with a $40°$ beam width. This causes the distance to objects to appear as hyperbolas [12], unless compensated for in the signal processing.

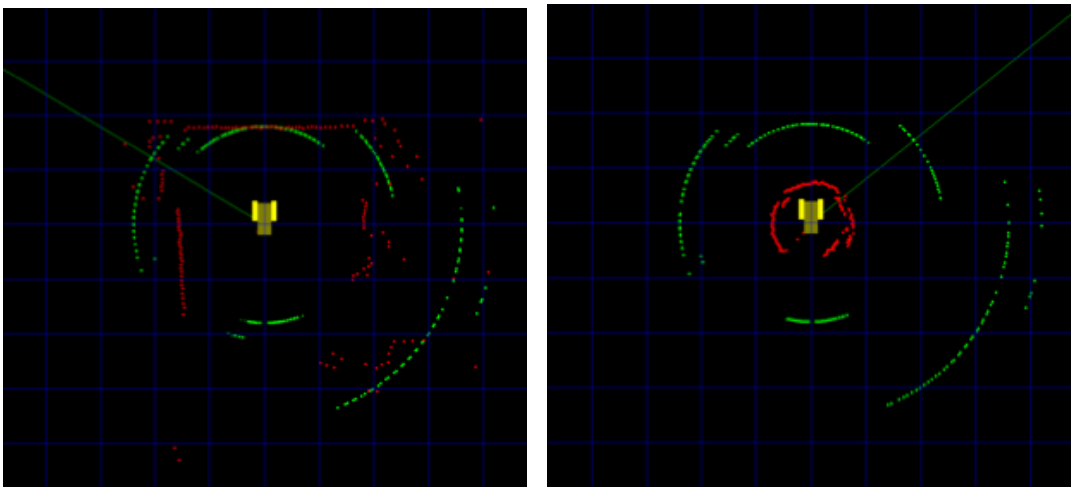
3.1.1 System properties

Principle of operation

UWBIR systems work by transmitting a pulse at a time t_0. When these EM waves hit an object, some of the energy in the pulse is reflected back towards the radar. After a certain time-delay, Δt, this energy can be sampled by the system. By knowing the speed of propagation of EM waves c, the distance d from the radar system to the object can be determined through

$$d = \frac{c\Delta t}{2}, \tag{3.1}$$

If we are able to program the time-delay Δt, we can choose which distances we want to look at. If, for example, we want to sample a reflection from anything potentially



(a) (b)

Figure 3.1: Distance returns from UWB (green) and LIDAR (red) distance sensors, (a): fog-free room, (b): fog-filled room. From [26].

located at a distance $d = 2$ m from the radar, we would choose the *sampling-delay*

$$\Delta t = \frac{2d}{c} = \frac{2 \cdot 2\text{m}}{3 \cdot 10^8 \text{m/s}} \approx 13.33\text{ns}. \tag{3.2}$$

Since these delays are so short, we are able to sample long distances within short periods of time.

Noise and clutter

The samples made by radar systems will always contain more information than only the reflections from the objects of interest. The aberrations contained in each sample can be split into two phenomenons:

- Clutter
- Noise

Clutter is the term used to describe the general unwanted reflections from the environment. E.g. if the sensor is faced towards a radiator mounted to a wall, the closest object to the sensor is the radiator and hence its reflection is the one we are primarily interested in. However, the wall behind it will also reflect a lot of the energy, and some of that reflected energy will again hit other objects on its way back to the receiving antenna. Therefore, the returned signal will be full of false positives, or positives from objects other than the one we are looking for. So-called 'ground clutter', i.e. unwanted reflections caused by energy bouncing of the floor, is one of the greatest sources of

such disturbances. *Noise* describes the rest of the deviations from the actual signal, such as thermal, atmospheric and electrical noise. Much of it is statistically white noise uncorrelated with the actual signal that can be effectively filtered, and is primarily caused by the transmitted waves bouncing off many reflectors smaller than the sensor's resolution.

A typical antenna setup in an UWBIR system consists of two antennas mounted next to each other, where one antenna transmits pulses using the programmed sampling-delays and the other samples their reflections. This causes a type of clutter referred to as antenna crosstalk, which is the power 'leakage' observed by the receiving antenna when the other is transmitting pulses. This appears in the samples as a constant bias, identifiable by sampling an empty room where the walls are padded in such a way that there is nothing reflecting the energy transmitted by the sensor. This is referred to as an anechoic chamber.

System model

With the preceding properties in mind, it is useful to decompose the sampled signal in the following manner:

$$\underline{z} = \underline{z}_o + \underline{z}_c + \underline{z}_n, \tag{3.3}$$

where $\underline{z}_o$ is the part of the signal containing the reflection from the object of interest in the scene, $\underline{z}_c$ is the clutter in the signal and $\underline{z}_n$ is the noise. What the radar system seeks to accomplish is to minimize the contributions $\underline{z}_c$ and $\underline{z}_n$ and isolate $\underline{z}_o$. In the following sections, methods of doing this will be reviewed.

3.1.2 Object penetration

UWBIR systems use much greater bandwidths than traditional radar does. The low parts of the frequency spectrum cause the transmitted signal to partially contain long wavelengths that can propagate through e.g. walls. This can present a challenge when mapping the surroundings of a robot if an object, or in fact the material the object is composed of, reflects too little energy to register in the receiving antenna. On the other hand, it has also enabled applications like e.g. through-wall imaging or ground-penetrating land mine detectors to emerge.

3.2 UWBIR data processing

Central in this thesis will be filtering of raw UWBIR data in order to properly extract the objects present in the frame. This section presents the algorithms that will be tested in later chapters.

First a note on terminology: the word *frame* will be used to describe a full measurement from an UWBIR system, i.e. the raw data returned by requesting a single measurement. The word *sample*, when referring to the UWBIR system, will be used to

describe the individual data points one frame consists of. However, *sampled* will still mean that a measurement, i.e. frame, has been made.

Figure 3.2 shows one frame with (particularly noisy) raw data from the UWBIR system that will be used in this thesis. The frame consists of 1024 samples, representing a 4 m sensing range with a 10 cm initial offset (i.e. sample 0 represents the energy reflected from the point located 10 cm from the receiving antenna, and sample 1024 contains the energy from 410 cm away). These samples are acquired by adjusting the sampling-delays as introduced in Section 3.1.1. The reflection from the main object in the scene is the 'peak' around sample 240; all the peaks before that are noise and/or clutter. In this state, the data from the system is almost unusable, and motivates putting effort into filtering.

UWBIR systems are so-called multiple-echo systems; meaning that the entire spatial range of the sensor is sampled and higher amplitude is indicated in the samples corresponding to the area the objects were located in. This is one of the advantages UWBIR has over laser range finders, which only return one distance. A secondary object can barely be recognized in the raw data frame in Figure 3.2, around sample 600. This corresponds to a drywall located behind the main object, which is a low brick wall.

The following subsections will review three methods of processing raw UWBIR data, which can be used separately or in combination: *singular value decomposition*, *clutter map* and *range compensation*.

3.2.1 Singular value decomposition

SVD, or *singular value decomposition*, is a method for factorizing rectangular matrices, and is a form of eigenanalysis [17]. For UWBIR data, it is a powerful algorithm for removing clutter. In Chapter 2 the set of measurements Z was introduced. In that chapter the set consisted of measured distances; here these distances are replaced with the full vectors of raw data,

$$Z = \begin{bmatrix} \underline{z}_1 \\ \underline{z}_2 \\ ... \\ \underline{z}_m \end{bmatrix}, \tag{3.4}$$

where $z_m = [d_1, d_2, ..., d_{1024}]$. As examined in Section 3.1.1, the observations can be split into subsets containing the reflections of the objects, clutter and noise. We can then use SVD to factorize the matrix with the set of raw data measurements of rank r, so that

$$Z = USV^H, \tag{3.5}$$

where U and V are unitary matrices of rank $m \times m$ and 1024×1024, and S is a matrix containing the r eigenvalues of Z, sorted in descending order. Using the statistical properties of clutter, we assume that the eigenvector with the greatest eigenvalue σ_1 is the vector describing the direction of the clutter, $|\underline{z}_c|$, and hence nulling all contributions from this direction removes much of the clutter in the measurements.

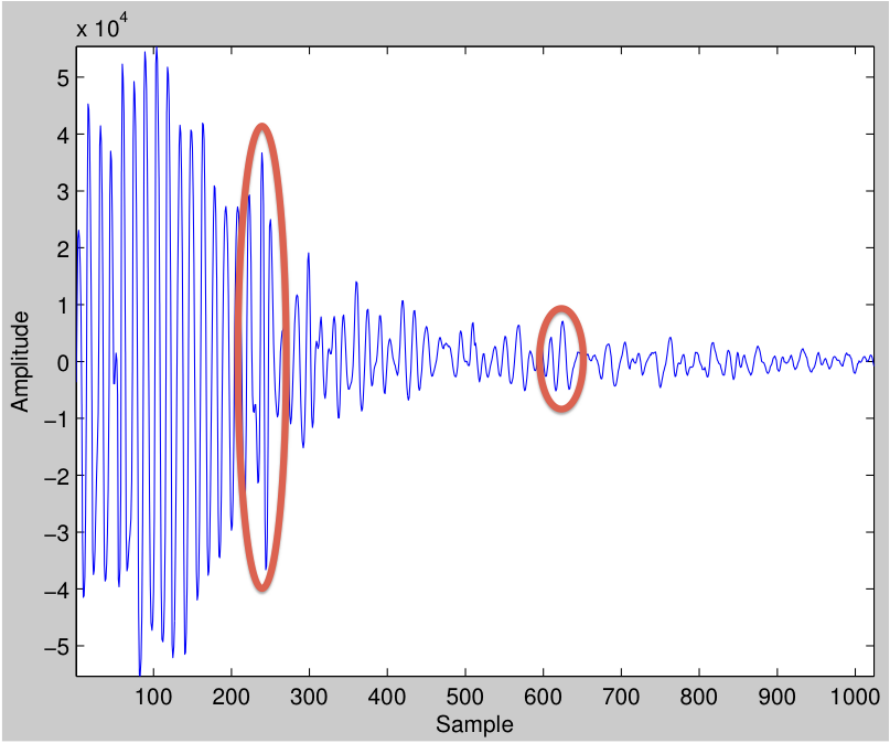


Figure 3.2: Frame of raw data. The high amplitudes at the start of the frame are caused by significant amounts of noise and clutter. Objects are highlighted.

Figure 3.3 shows the frame from Figure 3.2 after clutter has been removed by means of SVD. The frame is hardly recognizable anymore, and the main object in the frame can clearly be seen.

The SVD is a very good clutter removal scheme, but it requires that the data is already collected, as well as being a very time-consuming algorithm. Hence it can hardly be performed real-time to its fullest extent, and is therefore possibly unfit for most SLAM purposes. If these challenges can be overcome, however, there are few methods that surpass SVD in performance.

3.2.2 Clutter map

One of the most common and effective ways of suppressing clutter in radar data is filtering by using a *clutter map* [18]. These are often split into two different kinds: *static* and *adaptive*. A static clutter map can be obtained by sampling an anechoic chamber. The only peaks that are present in the resulting sample are caused by antenna crosstalk and potentially self-noise. This sample can be referred to as a static clutter map, because it describes only clutter and no signal. It can therefore be subtracted from all frames that contain an actual signal in order to suppress much of the clutter in the reflection.

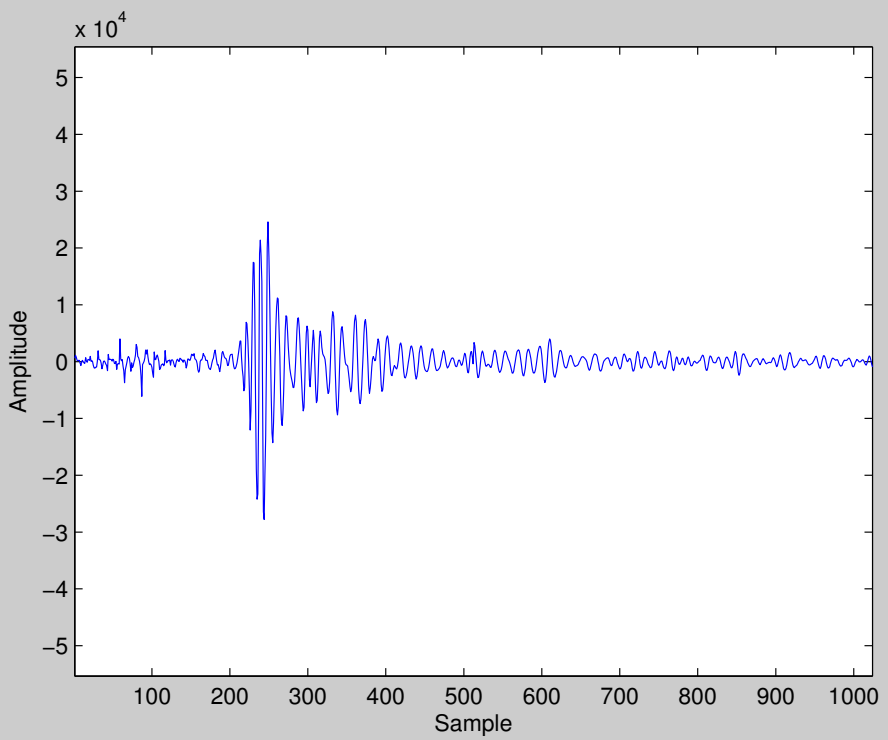


Figure 3.3: Frame of data after clutter removal using SVD.

An adaptive clutter map can be realized by

$$C_i = w \cdot X_i + (1 - w) \cdot C_{i-1}, \tag{3.6}$$

where C_i is the clutter map and X_i is the sensor sample at time-instant i, and w is a weight factor determining the 'speed' of the clutter map. If w is small, much of the data in the new sample is preserved. By updating C every time a new frame is obtained, the 'static' parts in the frames will be described by C so that the parts that varies from frame to frame (believed to be the signal with the actual reflection) will be kept in X_i when subtracting the clutter map from the current frame.

The adaptive clutter map has the drawback that it requires some calibration in order to be effective. Calibration is the process of adapting the clutter map to the current scene, by successfully identifying the static objects present. This can be achieved by e.g. processing continuous samples of a scene where a single static object is recorded by a moving sensor. If properly calibrated, the adaptive clutter map can potentially be a more effective filter than the static version because it adapts to the clutter generated by the scene and not just the clutter generated by the sensor itself. Figure 3.4 shows the frame after being filtered through an adaptive, calibrated clutter map. The main object is clearly identifiable, although the amplitude of the reflection is greatly reduced from the original, but the secondary object has been removed as clutter.

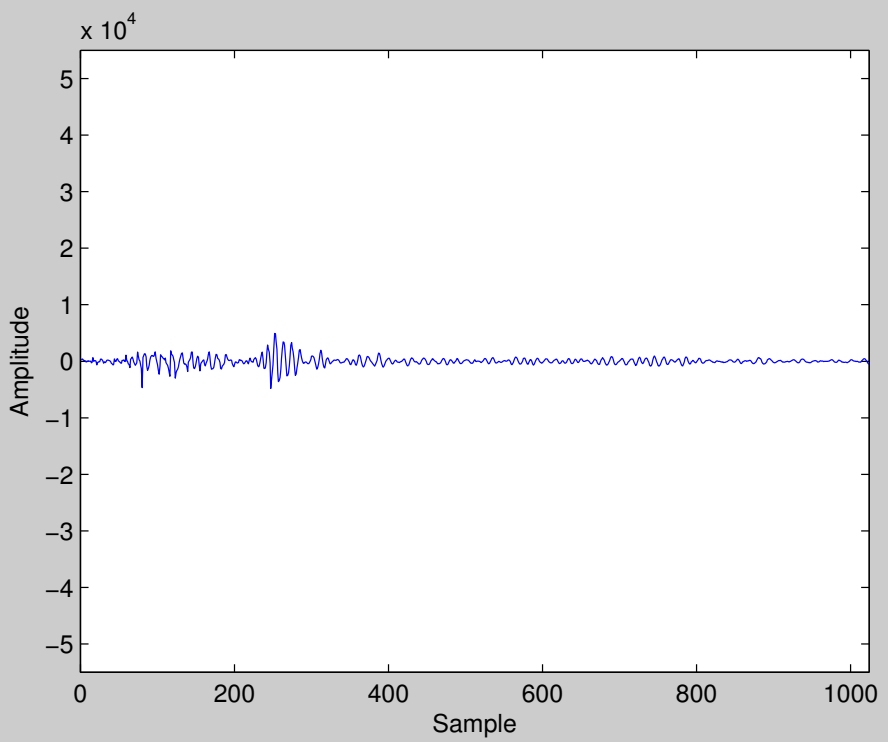


Figure 3.4: Frame of data after clutter removal using a calibrated adaptive clutter map.

3.2.3 Range compensation

Theory states that all propagating waves will decrease in amplitude, caused by physical effects such as dissipation, dispersion and attenuation [17], and an UWBIR system is no exception. How great these effects are depends on system-specific factors such as transmitted frequency and beam pattern, as well as environment-specific factors like temperature. For short ranges the effects can often be ignored, as most objects of interest will still have a reflection large enough to be recognized. However, with applications such as through-wall sensing and object recognition by means of amplitude magnitude (which will be discussed in the next chapter), compensating for wave deterioration can be a critical part of the signal processing.

4 Data association with UWBIR

In this chapter, methods and algorithms for object recognition will be introduced, with the focus being on UWBIR data.

4.1 Objects in UWBIR data

As previously stated, data association is one of the biggest challenges in SLAM. UWBIR systems show great promise with regards to identifying known objects, due to properties of the transmitted and returned EM signals.

4.1.1 Signal strength

A very straight-forward way of classifying objects within a returned signal is to look at the magnitude of the reflection, i.e. classify by the *reflectivity* of the object. Different materials have different absorption-rate of EM-waves, and this is an important factor in how much of the energy of the transmitted signal that is returned to the receiving antenna. Another factor is the *geometry* of the object and the angle with which the waves hit it; corners for instance, as shown in Figure 4.1, reflect much of the energy back to the source.

Using these two properties, one should be able to separate e.g. metal objects from wooden, and corners from walls. The latter in particular is very useful in SLAM, as edge-detection is a vital part in successfully identifying landmarks.

4.1.2 Pulse signature

Figure 4.2 shows the UWB signal which is returned from (a) a metal plate and (b) a wooden plate. This is a different system from the one reviewed in Section 3.2; the structure of the returned signal is the same, but this system uses more samples per frame. The amplitude of the signal reflected from the metal plate is approximately three times greater than the signal reflected from the wooden plate. There are, however, more that can be learned from evaluating the returned signal. In [23] it is shown that due to the high resolution of the UWB pulse, it is possible to identify what kind of objects are present in the sample by evaluating the *pulse signature* of the returned signal. The pulse signature represents the deformation the transmitted pulse is subject to when

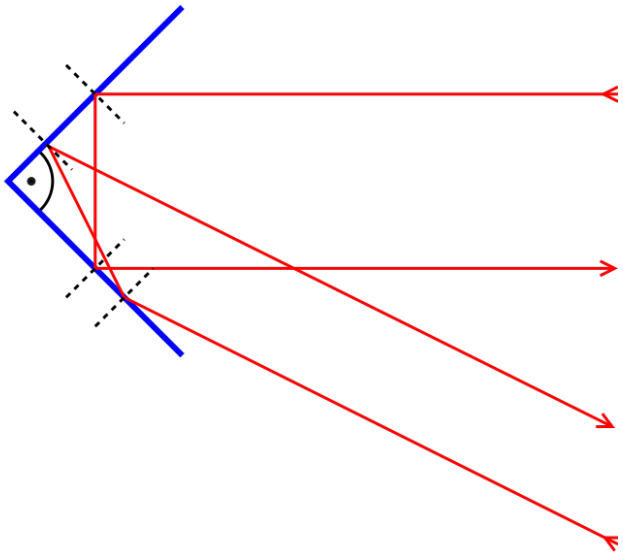
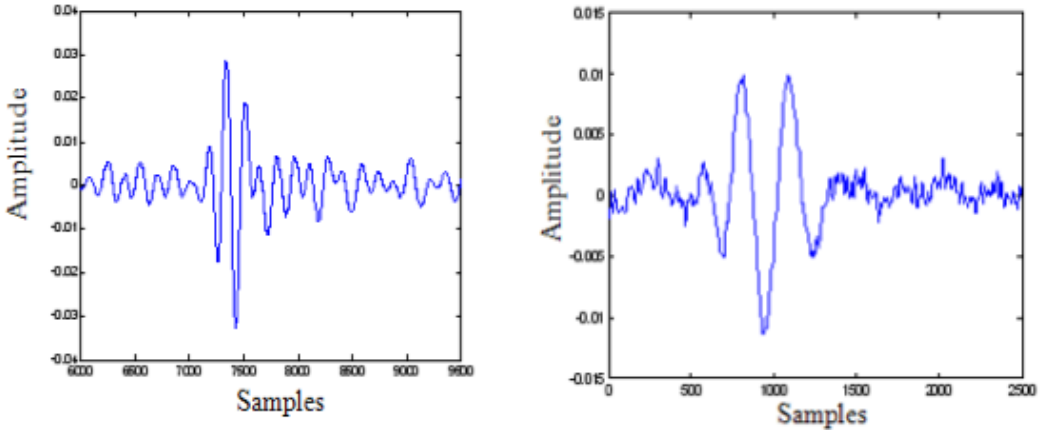


Figure 4.1: Corner reflector.

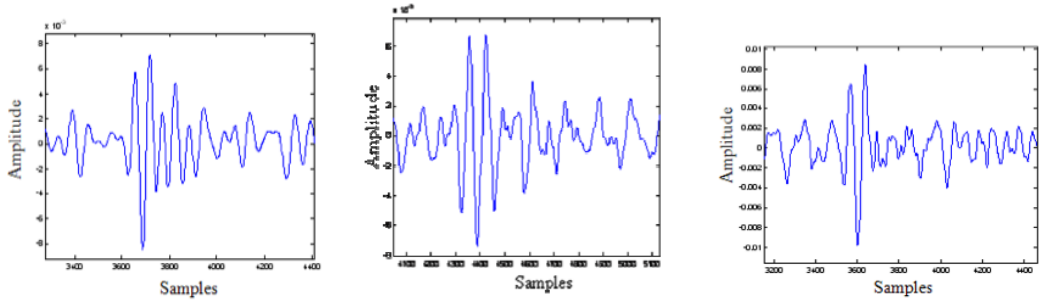
reflecting off an object. The identification consists of performing a correlation analysis between the received signal and a number of known pulse signatures. In order for this to work all the signals must be normalized, as the amplitude of the returned signal is proportional to the distance to the actual object within the sensor's range. The spatial extent of the reflection in samples, however, is the same no matter the range. This means that target recognition by pulse signature must be an operation performed separately to classification by signal strength as discussed in the previous subsection, since that method works by evaluating the relationship between displacement in the sample and amplitude. In addition, a returned signal will in most cases contain reflections from several objects besides the zero-mean Gaussian distributed noise as assumed by the EKF. This means that the sample must be divided into separate parts so that the individual reflections in the sample can be evaluated separately. This can potentially add a lot of computational effort to the SLAM algorithm.

If the problem of computational complexity can be solved, however, target recognition by pulse signature should be very helpful in successfully performing data association in SLAM. Figure 4.3 shows the reflected signals from a car, a motorway barrier and a pedestrian, which in [23] have all been successfully identified by solely evaluating individual pulse signatures within the same sample. In a SLAM setting, where landmarks are initially selected if they are present and identifiable in N samples, the ability to classify an object with the level of certainty that this method shows would be very valuable.



(a) (b)

Figure 4.2: Returned UWB signals from (a): metal plate, (b): wooden plate. From [23].



(a) (b) (c)

Figure 4.3: Returned UWB signals from (a): car, (b): motorway barrier, (c): pedestrian. From [23].

4.1.3 Uses in SLAM

Object classification of this kind is not an essential part of SLAM, but theoretically it has many uses. Firstly, it is useful in the SLAM-product (the map) in itself. Imagine that the robot performing the SLAM is sent in to investigate a burning building, where the smoke is obstructing the sight of the firefighters. The robot can then potentially report if there is a human present in the environment, who might be passed out from inhaling smoke and is not able to alert the firefighters of her presence herself. Similarly, if the robot can report if a wall is made of bricks or sheetrock, it can help the firefighters determine if it is weight-bearing and hence assess the dangers in the burning building. In a military setting, it can be used for locating houses or even persons of interest in a hostile environment.

Secondly, classification of this kind can be useful in improving the accuracy of the

SLAM maps. As discussed in Section 2.2, one of the greatest difficulties in SLAM is data association, i.e. recognizing previously observed landmarks. If corners can be detected, or a metal radiator can be separated from the brick wall it is attached to, these properties can potentially be used in improving the hit-rate of the data associations, hence making the mapping more accurate.

4.1.4 Linear regression

Central in the loop-closure problem as discussed in Section 2.3.2 is the ability to perform data association, especially where walls and corners are concerned. A very common method of extracting these types of features is *linear regression*, i.e. combine several measurements to form lines. Using the notation introduced in Chapter 2, after a distance measurement Z is obtained, it is combined with the current estimated state X to translate the points from the local coordinate system of the robot to the global map of the environment; i.e. produce absolute measurement coordinates

$$\underline{x}_m, \underline{y}_m \in m(X, Z). \tag{4.1}$$

Linear regression is then to find the line that best fits these measurements [8],

$$\underline{y} = \alpha \underline{x}_m + \beta. \tag{4.2}$$

A standard estimate of the linear regression is then the least-squares algorithm,

$$\hat{\alpha} = \frac{\mathrm{Cov}[\underline{x}_m, \underline{y}_m]}{\mathrm{Var}[\underline{x}_m]} = \frac{\overline{\underline{x}_m \underline{y}_m} - \overline{\underline{x}_m}\,\overline{\underline{y}_m}}{\overline{\underline{x}_m^2} - \overline{\underline{x}_m}^2}, \tag{4.3}$$

and

$$\hat{\beta} = \overline{\underline{y}_m} - \hat{\alpha}\overline{\underline{x}_m}. \tag{4.4}$$

The quality of the line therefore depends on the correct data points being selected for combination. This process can potentially by aided by using classification algorithms to determine the material of the object.

4.2 Classification algorithms

There are many families of algorithms for performing object classification; some are optimized for speed, some for accuracy, and some works only for a specific type of data input. Here we will look at some different algorithms and test how well they are able to classify UWBIR raw data. The theory of the algorithms is, unless stated otherwise, described in greater detail in [20].

In order to keep the computational effort needed at a minimum the classification can be performed by using e.g. a trained neural network. This has the advantage of being a fast classifier, but some a priori knowledge of the robot's environment is needed,

since the network must be trained on objects that are believed to be present and time-invariant in the environment. If no such knowledge is present, other classifiers can be used instead such as the k-means algorithm. The algorithms introduced below are selected for testing mostly because of these different properties.

There are three main types of classification algorithms: *supervised*, *unsupervised* and *reinforcement* learning. In this thesis, supervised and unsupervised classifiers will be implemented, and are described in greater detail in the next sections. Reinforcement learning is in a way a hybrid between the two other, and is traditionally used in control theory or to move a robot in a 'sensible' way in an unknown environment. Its relevance in this thesis was therefore considered to be minimal compared to supervised and unsupervised learning.

4.2.1 Supervised learning

Supervised learning algorithms are a family of algorithms that relies on a priori information about the environment. The training epoch consist of using pre-collected data where the classes already are known. When the algorithm generates an output, the error between the output and the desired result can be calculated and used to correct the model, hence improving the performance in the next iteration.

If trained for too many iterations, or in the case of training data with little variation, supervised learners run the risk of becoming *overfitted*. This means that they have adapted too well to the training data and lose their generality. Instead of identifying the prime general components by which it is possible to separate the different classes from each other, the algorithm learns every little variation in the training data and is not able to classify anything other than this precise data. This problem is illustrated in Figure 4.4, where the red and blue dots represent input nodes of two different classes in some training data. The black parabola is the ideal separator between them, but due to overfitting the learner has instead generated a polynomial of a much higher degree to use as a separator. This separator will clearly perform much worse when the algorithm is subjected to new data.

k Nearest Neighbors

Nearest neighbor searches, in this case k nearest neighbors (kNN), are one of the simplest yet most robust form of classifiers. It classifies data by calculating the distance between the input to be classified and some labelled reference frames in the input space. For UWBIR data, the distances d can be found using the location of the samples in the euclidean space, i.e.

$$d = \sqrt{\sum_{i=0}^{n} (\underline{z}_i - \underline{t}_i)^2}, \tag{4.5}$$

where n is the number of samples in each frame, $\underline{z}$ is the frame to be classified and $\underline{t}$ is the reference frame. The k closest neighbors are found, and the most frequent of the classes of these k is set as the class of frame $\underline{z}$.

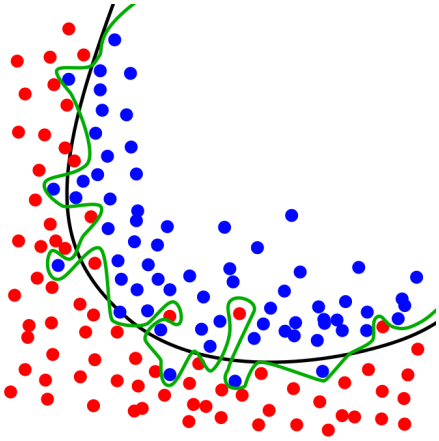


Figure 4.4: A supervised learning algorithm that has become overfitted: the generality of the input classes has been lost as the algorithm has adapted to the outliers in the collection.

kNN needs no training before it can be used, but as can be seen from equation 4.5, it needs many computations to classify the data during execution. Each of the N data points must be compared to all the other data points in the same position in the reference database consisting of M samples, making it an $\mathcal{O}(NM^2)$ operation.

Multi-Layer Perceptron

The multi-layer perceptron (MLP) classifier is one of the algorithms that attempts to recognize objects by imitating the human brain. This is done by constructing a neural network, which is illustrated in Figure 4.5. The nodes in the input layer correspond to their respective value in the input data. These inputs are fed forwards towards the output layer, through one or more hidden layers. Each input node is connected to all nodes in the first hidden layer. All of these nodes are connected to all the nodes in the next layer, and so on. In a 'real' neural network, e.g. the brain, the nodes in the network triggers depending on the input they receive measured against a threshold. In MLP, however, the nodes instead obtain a value based on the input and a sigmoid function, which has the advantage of being close in shape to a threshold function, whilst still being continuous. This is illustrated in Figure 4.6. The most common sigmoid function used as the activation function is

$$a = \frac{1}{1 + e^{-\beta h}}, \tag{4.6}$$

where a is the value of the neuron, h is the input value to the neuron and β is a positive parameter, set before starting the algorithm and can not be changed during its execution.

The connection between each pair of nodes is individually weighted, with a value in the range $[0...1]$. Thus, the combination of the input, the weights of the connections

and the sigmoid function decides the value for the current node. This value contributes to calculating the value of the nodes in the next layer, and so on until the output layer is reached. The output nodes finally form the output of the algorithm.

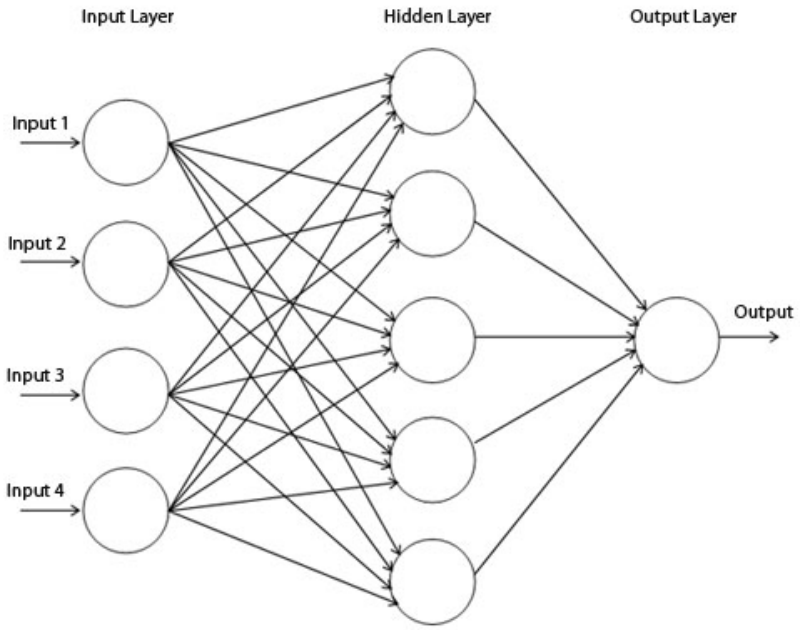


Figure 4.5: Multi-layer perceptron network.

The MLP learns by calculating the sum-of-squares error e between the output (z) and the desired output (t) for each node, that is

$$e_k = z_k - t_k, \tag{4.7}$$

and

$$e = 1\frac{1}{2}\sum_{k=1}^{n} e_k^2, \tag{4.8}$$

where w are the weights and n the number of output nodes in the network. The error is then backpropagated through the layers, i.e. used to update the weights in the network.

Training the MLP is a time-consuming task, depending on the number of data points, hidden layers, etc. Using a pre-trained neural network, however, only has a complexity of $\mathcal{O}(MN)$, where M is the number of nodes in the network and N is the number of data points in the samples. It is still not a fast algorithm, but it is a lot better than kNN.

MLP is one of the most used algorithms for classification to date, both because of its speed and accuracy. The main drawback of using it is, as with the kNN-algorithm, that some a priori knowledge about the environment is needed.

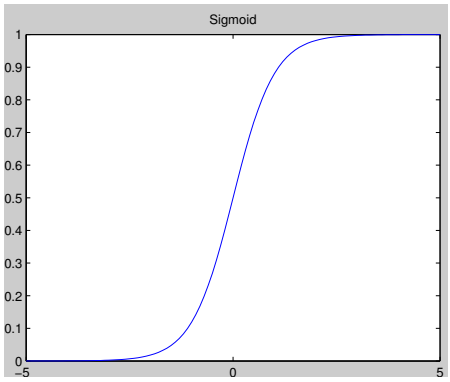


Figure 4.6: Sigmoid function commonly used as an activation function in MLP.

Support Vector Machine

All input nodes are defined in an input space, given by the number of inputs in the data. In order to be able to classify the nodes, they have to be *separable* in the input space, meaning that the nodes of the same class have to be able to be grouped and become distinguishable from the groups of the other classes. If the groups are separable by drawing a single line or plane in the input space, the groups are *linearly separable*, and relatively easy to classify. However, few real world problems are linearly separable. The MLP reviewed above separates the groups by using complex separators, i.e. non-linear. Support vector machines (SVM) instead use a *kernel function* to map the input data to a higher dimension, until it becomes linearly separable. This is illustrated i Figure 4.7 where the input nodes of classes red and blue are originally defined only in one dimension, and can not be separated by drawing a straight line. When mapped to two dimensions, however, they become linearly separable.

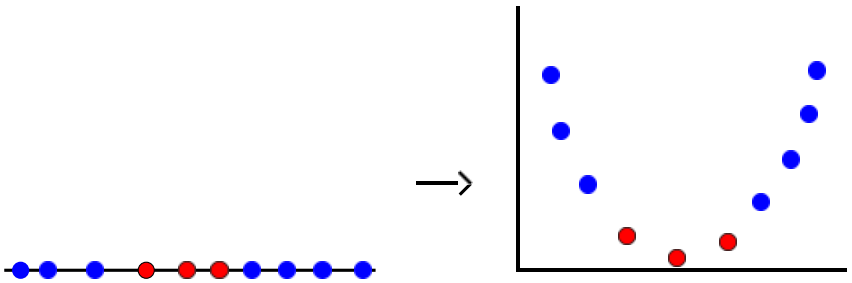


Figure 4.7: Nodes in an input space being mapped to a higher dimension, in order to become linearly separable.

The kernel function is not a fixed expression; there are many variants, and one must be chosen depending on the input data. Typically, the kernel function involves the dot product of the *feature vectors* in the input, i.e. the different nodes in their original

dimensionality.

When linear separability has been achieved, the term 'support vector' can be defined as the line, plane, etc. that separates two classes in such a way that the separator is located in the middle of the space between the classes. This is referred to as being the *maximum margin hyperplane*. This is trivial to find with linear separability, and the motivation behind finding it is to minimize the probability of misclassification.

The SVM is one of the most applicable and robust classifiers, and it performs very well in both speed and accuracy compared to other algorithms.

4.2.2 Unsupervised learning

Opposite of the supervised learning algorithms, unsupervised learning algorithms require no a priori information about the environment. Instead of evaluating a node in the input space against an existing database with known classes, the nodes are compared only to each other, and clustered based on their similarities. In this thesis, the *k-means* algorithm will be evaluated.

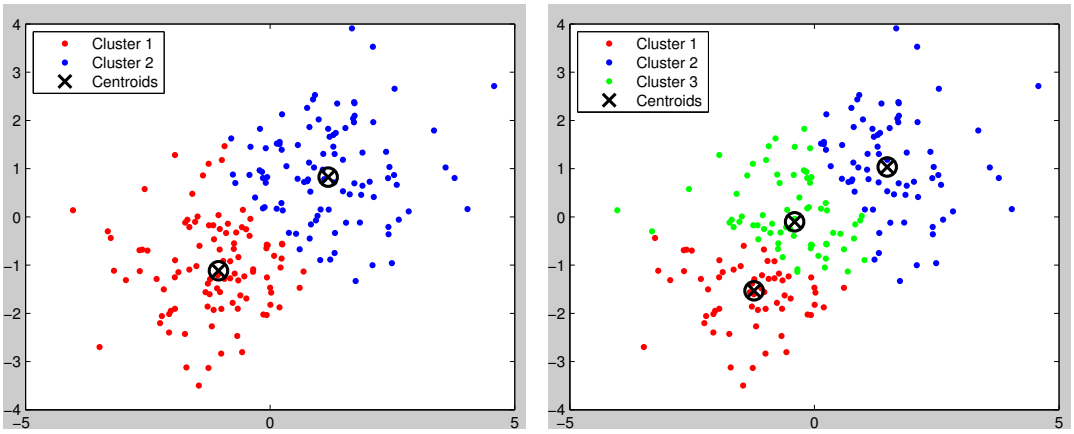
k-means

The k-means algorithm is similar to kNN, in that it uses separation in the form of Euclidean distance in the input space to determine class. The main difference is that k-means is an unsupervised learning scheme. In short, this means that while kNN finds the closest match in the database and assigns that class to the sample, k-means creates its own classes, or *clusters*. At the outset of the algorithm, k positions in the input space are chosen at random, and are assigned individual clusters; they become *cluster centers*. Each node in the input space is then evaluated (i.e. separation in Euclidean terms is calculated) against all the cluster centers, and is assigned the class belonging to the closest center. When all the nodes are clustered in this manner, the center of each cluster is moved to the mean position of all the nodes assigned to it, hence altering the distances previously used to assign classes. Thus, each node must again be evaluated against all the cluster centers and possibly be re-assigned a cluster. This process loops until the cluster centers stops moving, or some other termination criterion is met.

Figure 4.8 shows how a k-means-algorithm has clustered an array with random data, for different values of k. Depending on the randomly generated start-position of the cluster centers and the geography of the nodes in the input space, the algorithm can yield different output from execution to execution.

k-means is possibly quite well suited to classifying UWBIR data, since the pulse signatures are prone to appear with some variations depending on the angle of inclination etc., as previously discussed. With this scheme, we are able to adapt the database to the current environment and not just the ideal samples that are present in the original signature database. The danger is that we are very prone to outliers, which potentially can shift the cluster centers long distances at a time and corrupt the entire database.

The complexity and computational effort required to classify using k-means depends on the number of clusters and how many iterations are spent on moving the cluster centers when new data is obtained.



(a) (b)

Figure 4.8: k-means-generated clusters with randomly generated input data, (a) k=2, (b) k=3.

Part III

Implementations

5 Setup of experiments

This chapter describes the experiments performed in this thesis. This includes hardware setup, software utilized and data recording schemes.

Central in this thesis has been practical work, i.e. the designing and building of a mobile robot capable of collecting the data needed for performing SLAM in a realistic way. The processes involved in implementing the robot sensing system presented in this chapter has taken up more than half of the time spent on this thesis. Examples of especially challenging tasks were selecting and programming the micro computer for the robot, and obtaining accurate positional data from the wheels of the robot. This will be examined in greater detail in the separate sections in this chapter, but little is said of the processes of creating the robot. This is because even though it was a very big part of the performed work, the platform used for collecting the raw radar data is still only a platform, and it is the data processing made possible by the platform that is central in this thesis.

5.1 Goal of experiments

The goal of this thesis is not to develop the SLAM algorithm in itself, but investigate how well UWBIR based sensors perform when put to such a use. Therefore, many different combinations of signal processing and classification algorithms combined with different SLAM-algorithms will be implemented. Hence it is not an efficient approach to perform the experiments in real-time, but rather record the data needed at the outset of the experiments and perform offline processing afterwards. The constraint of real-time that SLAM has will not be ignored, but the focus will be on optimizing the quality of the mapping and localization.

5.2 Recording data

A radar sensor mounted on a mobile platform will record the data. This will be described in greater detail in the next section. Instead of autonomous behavior, the robot will be driven 'manually' in the environment by steering implemented in Matlab-scripts. This is not as good as full autonomy, but the collected data will have full validity and is hence preferable to simulated data.

The robot will use a 'stop-and-shoot' sampling scheme. This means that it will drive for a given distance, then come to a full stop and sample its surroundings, before starting to move again. For this thesis the robot will mainly be programmed to drive 60cm for each interval, and sample with a field-of-view of 180°. This is realized by panning the sensor from -90° to 90°, with 0° being straight ahead of the robot. Further details follows in sections 5.3.1 and 5.4.2.

Once the robot has collected all the data, the data will be imported into Matlab for processing.

5.3 The robot

This section describes the robot and its components. Figure 5.1 shows an overview of the structure of the robot. It consists of a base platform movable by four servo-controlled wheels, and the servos controlling the two front wheels are equipped with rotary encoders. These work by counting the number of revolutions the servos make. Further, a secondary platform is attached to the base platform, made pannable by a stepper motor. Mounted on this secondary platform is an UWBIR distance sensor. Controlling all these components is a micro computer, which serves as the main controller. The servos, stepper motor and rotary encoders are connected to the controller through a custom designed connection board, and the UWBIR sensor is connected through USB. The micro computer itself is controlled over Wifi by Matlab scripts.

The rest of this section will examine the individual components in detail.

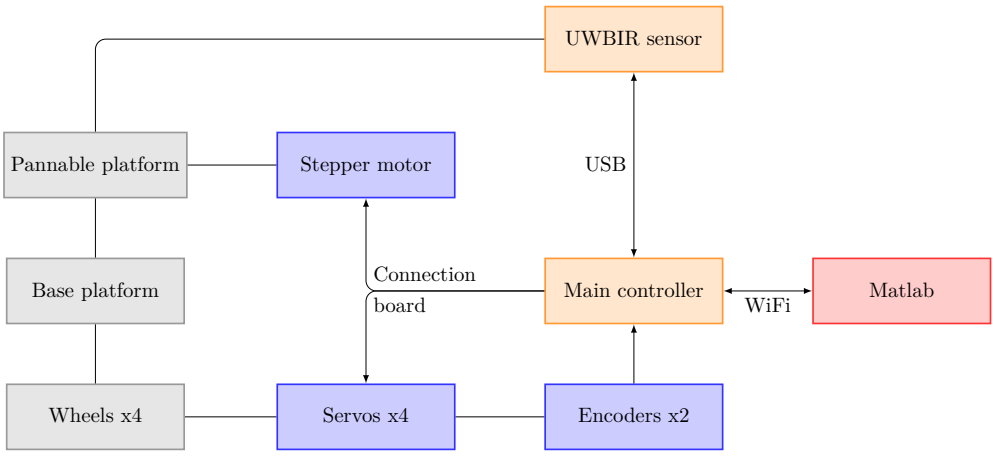


Figure 5.1: Block schematic of the robot and its components. The lines represent mechanical links while the arrows are electrical connections, indicating the direction of the data transactions.

5.3.1 D1S UWBIR sensor

The sensor used in the experiments is the D1S radar sensor from Intelligent Agent AS [16]. It is one of the first distance sensors to use UWBIR technology, and has three modes of operation:

- Transmit distance measurements
- Transmit raw radar data
- Transmit processed raw data

It has a maximum sensing range of 4m, with a range resolution (x-axis in Figure 5.2b) of 4mm. The transmitted beam is ~40° in both the horizontal and vertical plane. This means that the resolution in azimuth-direction (y-axis in Figure 5.2b) is poor; the distances extracted from a frame is assumed to be located in the center of the beam to minimize the potential error. Additionally, the reported distances can be ambiguous when panning the sensor. Therefore, the full 180° pan of the robot will be split into only 16 steps, so the sensor rotates 11.25° at a time, giving the field-of-view as shown in Figure 5.3.

The sensor is optimized for providing continuous distance measurements of a scene. This implies that it adapts to the scene it is observing and only yields results with full accuracy if it is allowed time to calibrate its internal clutter map (as discussed in Section 3.2.2) to the scene. The stop-and-shoot sampling scheme is therefore not likely to provide measurements with a high enough accuracy, so the sensor will be programmed to also transmit its raw radar data in addition to the distances whilst sampling. The raw data will later be used to improve the extraction of features in the sensor's field-of-view.

5.3.2 Mobile platform

The D1S sensor is mounted on a pannable platform, which again is a part of a rover controlled by four servo motors [22], as shown in Figure 5.2. Rotary encoders are connected to the two front wheels. These are the proprioceptive sensors of the robot, and combined with rugged, high-traction wheels enables accurate estimates of the robot's pose and position, if the surface the robot is moving on is uncluttered and non-slippery.

Beaglebone

The brain of the robot is the Beaglebone development board [15]. The Beaglebone features plenty of general purpose input/output (*GPIO*) pins, and has a powerful micro controller from Texas Instruments. This enables digital signal processing real-time whilst recording input from both of the sensors and controlling all of the peripherals. This, in addition to being relatively low-cost and having both USB and ethernet interfaces, is the reason this particular platform was chosen.

(a) (b)

Figure 5.2: The robot; (a) photo of the robot, (b) illustration of the robot's coordinate system. The x-axis will be referred to as the range-direction of the robot, and the y-axis as the azimuth-direction.

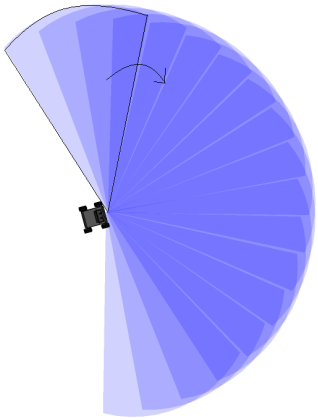


Figure 5.3: Field-of-view of D1S when panning 16 steps of 11.25° each.

Pan and servos

The robot is driven by four simple DC servo motors, controlled by the Beaglebone through a motor driver board. Similarly, a stepper motor controlled by the Beaglebone in the same manner powers the pannable platform the D1S is mounted on. The stepper motor was chosen over the DC servo because of the accuracy needed in the angle of rotation; the data used by the SLAM algorithms is calculated by using fixed angles, so it is essential for the results that these are very accurate.

Rotary encoders

The robot is equipped with two Hall-effect shaft encoders. These are the sole proprioceptive sensors used in this thesis, and their function is to 'encode' the rotation

Figure 5.4: The Beaglebone.

Figure 5.5: The servo motor with the rotary encoder attached.

of the wheels so that the movements of the robot can be extracted. The encoders are mechanically connected to the servos of the front wheels of the robot, and transmit pulses to the Beaglebone when a rotation greater than a given threshold occurs. The Beaglebone thus stores the incremental angle of rotation in radians. The encoders at hand yields 3 pulses per motor rotation, and the servos have a 30:1 gear reduction, giving a total of 90 pulses per revolution of the wheels. This implies a theoretical resolution in positional data of

$$x = \frac{\text{wheel circumference}}{\text{pulses per revolution}} = \frac{12\text{cm} \cdot \pi}{90} \approx 0.42\text{cm}. \tag{5.1}$$

The encoders only count servo rotation, so for the data to be correct a 1:1 connection between the servos and the wheels must be assumed; e.g. if a wheel starts spinning due to lack of friction, the encoders will still report movement.

SLAMbot

A custom connection board for the robot was designed specifically for this thesis, called SLAMbot, which is shown in figures 5.6 and 5.7. The full schematic can be found in appendix B.

In the first test-runs of the robot, when no such printed circuit board (*PCB*) existed, the robot would suddenly stop working, or leap into maximum speed and crash into the nearest wall. This was because of short-circuits in the Beaglebone as a result of wires carrying 5V or 12V touching onto the GPIO-pins meant for 3.3V logic. So it became apparent that for stability and ease-of-use, some time should be spent on making proper and robust connections for the hardware.

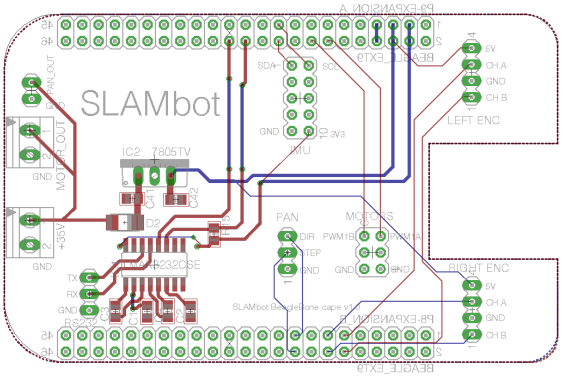


Figure 5.6: Connection board for the robot.

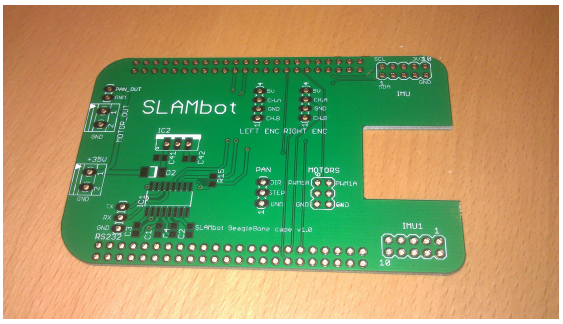


Figure 5.7: Printed connection board for the robot.

5.4 Environment

SLAM is used for both indoors and outdoors applications. This thesis will only examine indoor SLAM, as the sensor at hand has a range of only 4 meters. Outdoor terrain would also require another way of extracting the positional state of the robot, such as GPS,

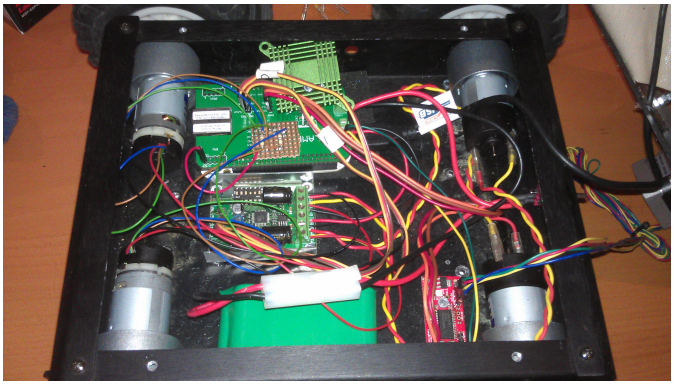


Figure 5.8: The insides of the robot.

since the rotary encoders on the wheels are only as accurate as the surface allows, as discussed in Section 5.3.2. Efforts should be made to examine the UWBIR technology for outdoor applications as well; this is conceptually no different from indoors.

5.4.1 Scene setups

Data was recorded from three different scenes in order to test the sensor in several settings. These are described in this section.

Scene 1: square walls

The first experiment will be set up with simple concrete blocks as walls, and the test will be simply to see how well these are depicted by the D1S sensor. The scene is shown in Figure 5.9. All vertical surfaces are spaced so that the robot never is farther than 4m from them. The walls are 60cm high. A simple scene such as this is included for calibration purposes only, i.e. to verify that the sensors are properly set up.

Scene 2: simple geometry

The second setup includes more materials than the first, and has more corners and varied geometry, as shown in Figure 5.10. The colors in the illustration show which materials are prevalent in the surfaces, however they are not completely accurate. This is because e.g. drywalls will often consist of a wooden inner structure as well as metal studs. Additionally, the wooden surface in the scene is simply tables with a metal structure lain down on the floor. These kinds of deviations are to some degree present in all surfaces.

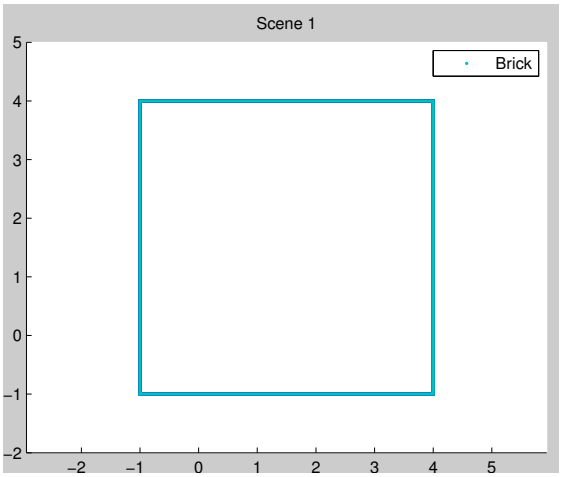


Figure 5.9: Illustration of scene 1. The colors show the prevalent material of the surface.

Scene 3: generic office

The third setup is a generic office environment, shown in Figure 5.11. Figure 5.12 shows the robot sampling the top-left corner of the scene. In addition to the surfaces shown, there were several objects present in the scene, such as chairs and tables in the lower left corner (whose metal legs are the only parts visible to the robot when driving on the floor). It is a scene with a much more complex geometry than the previous setups, as well as consisting of different kinds of materials with both large and small spacing. As with scene 2, the materials are all simply references. The upper right surface, for example, is in fact a kitchen wall with kitchen appliances, meaning that it is almost as much metal and drywall as it is wood. Additionally, the longest distance between two objects or boundaries in the scene exceeds the sensor's maximum range of 4m. It is therefore a scene well suited for analyzing the combination of SLAM and UWBIR.

5.4.2 Driving patterns

Positional data relying on only rotary encoders will quickly become inaccurate due to wheel drifting, etc. Therefore, for the sake of having as accurate positional data as possible, the robot drove through the environment with several different driving patterns. One of these patterns included hard-coded 90° turns, which proved to be the most accurate. The hard-coded turns caused only a minimal amount of drift on the wheels. Other, more natural driving patterns resulted in too inaccurate positional data to be usable for the purposes of this thesis.

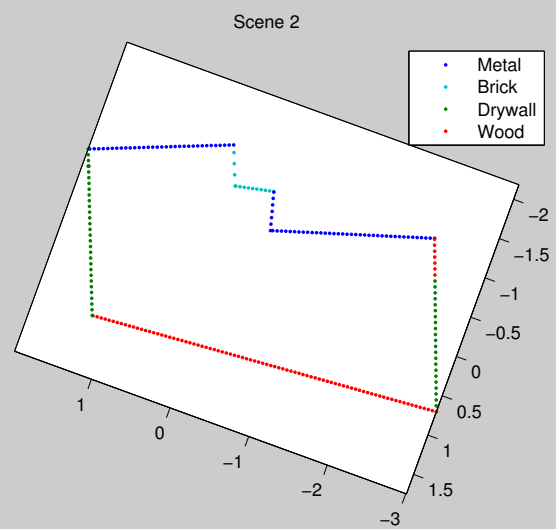


Figure 5.10: Illustration of scene 2. The colors show the prevalent material of the surface.

5.5 Software

Much different software has been developed and used in this thesis. This section briefly outlines the major components.

5.5.1 Robot firmware

The firmware controlling the robot is written in C and runs on an Angstrom Linux distribution. It communicates with the world via ethernet, using a simple protocol for sending and receiving commands:
`command#payload\n`
where `command` includes 'start-slam', 'turn-left' and others. `payload` can be degrees to turn, duration to drive forwards, etc.

5.5.2 Scripts

The scripts for communicating with the robot, as well as all signal processing algorithms and miscellaneous helper-functions are implemented in Matlab.

5.5.3 CAS toolbox

As previously stated, SLAM is a well established family of algorithms, and there are plenty of 'plug-and-play' implementations to be found. One of these is the CAS toolbox [2]. It is an implementation of SLAM for Matlab, designed to be adaptable to any type of sensor input. There is a default data parser for 2D Euclidean measurements, i.e. the measurements are given in (x,y)-coordinates relative to the robot. If the range data is

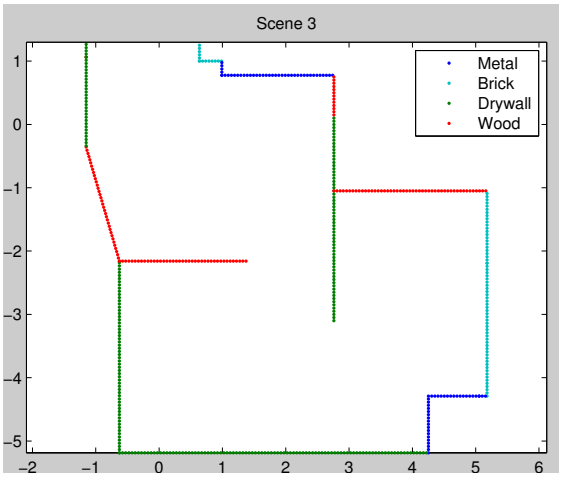


Figure 5.11: Illustration of scene 3. The colors show the prevalent material of the surface.

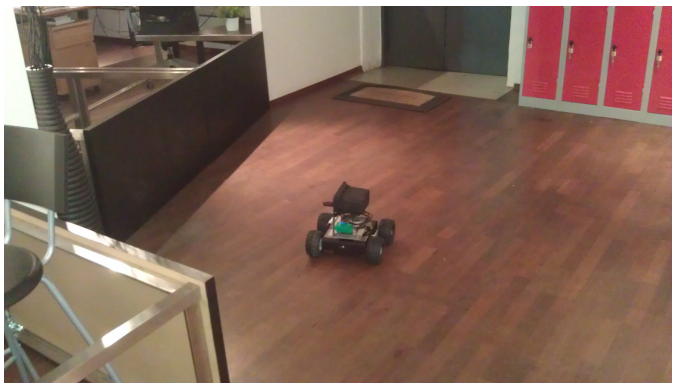


Figure 5.12: Picture of the robot sampling scene 3.

formatted differently, the user must write a specific parser. There is also a parser for proprioceptive sensor data from inertial measurement units (IMU), as well as rotary encoders.

The toolbox uses EKF to perform SLAM. This, along with the ease-of-use and the adaptability of the software, is the reason this particular toolbox will be used in this thesis. Measurements from an UWBIR sensor will typically be noisy, as was discussed in Chapter 3. Much of the noise is statistically white, which we know the EKF is very effective at removing. Further, the ability to do object recognition by analyzing the signal returns from the sensor can possibly be used to solve the loop-closure problem.

The CAS toolbox uses a line-based approach to extract features from the collection of measurements. In short, for each 'batch' of distance measurements (the 16 samples obtained per stop), the algorithm uses linear regression to attempt to fit the data points to line segments. The models used for this can be found in full in [3]. These line segments are combined to form complete lines which make up the navigational SLAM

map, as introduced in Section 2.5.

The CAS toolbox will be used as the main platform for experiments for the following reasons:

- It is implemented in Matlab, so it is easy to combine with the received data from the robot, as well as all signal processing algorithms
- It can draw raw data as well as processed SLAM, which is useful for testing several combinations and setups

Performance

In order to estimate how well the toolbox potentially can perform, the ground-truth models 5.10 and 5.11 was converted into simulation data in the same format as the actual experimental data will have. From this, the toolbox were able to generate the navigational maps shown in Figure 5.13. As can be observed, the toolbox completely extracts (almost) every line, given data points of both quantity and quality enough. Additionally, tweaking the parameters of the toolbox makes it able to extract every line, although at the cost of more false positives. The parameters include constants such as the size of the window used for 'remembering' old line segments and the significance level for matching line segments.

It is far from realistic to expect this level of accuracy when replacing the simulated data with the real data, because of the noise and other aberrations that will always be present when discretizing a continuous world. It is nevertheless useful to perform this kind of simulation in order to verify the validity of the software.

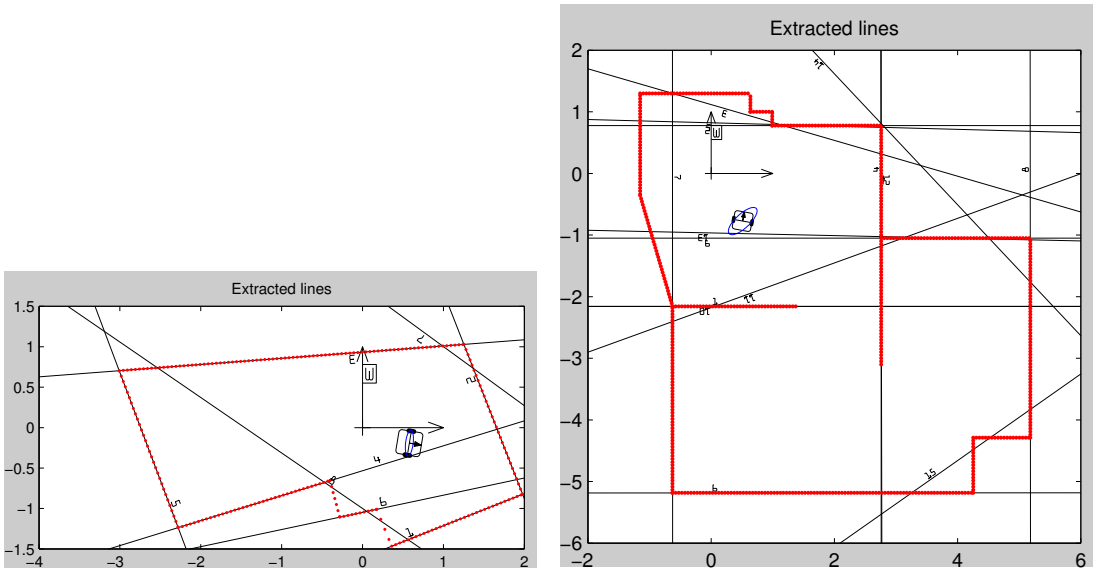
5.5.4 DP-SLAM

As with EKF-SLAM, there exists several SLAM-implementations based on the particle-filtering approach. One such is the distributed particle SLAM, or DP-SLAM [9].

The out-of-the-box state of DP-SLAM is very similar to the CAS toolbox, in that it is created for interpreting data from laser range scanners. However, it takes only complete sets of odometry data as input, as opposed to the CAS toolbox that estimates these from raw rotary encoder data. For ease-of-use, the odometry data will be acquired through the CAS toolbox, since the mechanisms already are in place.

As opposed to most other SLAM algorithms, DP-SLAM does not extract landmarks during execution. Landmarks are especially used for closing loops and fine-tuning the generated map, but DP-SLAM has a good enough accuracy to not need any loop-closing techniques [9].

The combinations of raw data filtering and classification-data yielding the seemingly best results using the CAS toolbox will be tested in DP-SLAM.



(a) (b)

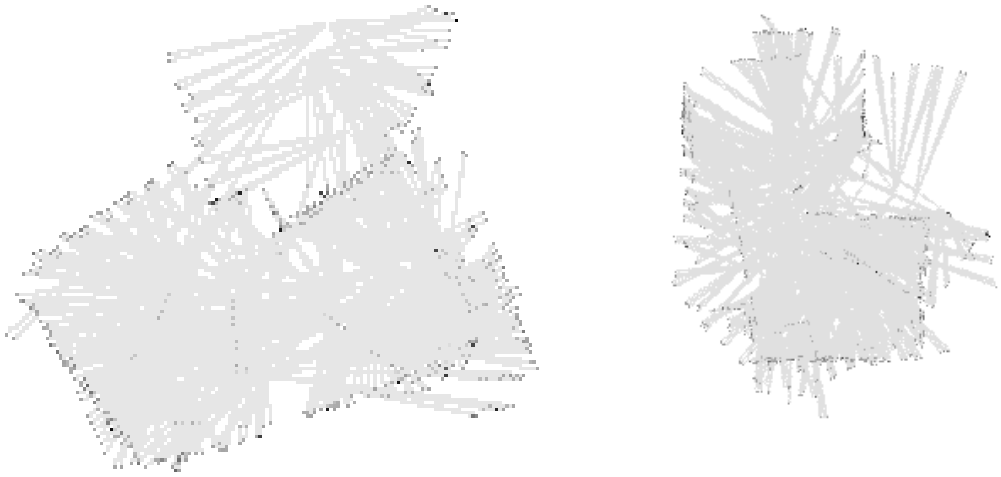
Figure 5.13: The lines the CAS toolbox were able to extract from the model of (a) scene 2, (b) scene 3.

Performance

Of the same reasons as given in Section 5.5.3, simulated data was used to evaluate the performance of DP-SLAM. The resulting maps are shown in Figure 5.14. As can be observed, scene 2 is very accurately reproduced, with a few outliers. Scene 3 is also highly recognizable, but there are a higher amount of outliers present, and some measurements seem misaligned. It is likely that an even higher amount of samples in the simulated data would have increased the accuracy of the maps further, as there are some blank spots in the maps. Additionally, there are a number of internal parameters in the algorithm for specifying such things as the motion model of the robot, and these have not been optimized for the data collected in this thesis.

5.5.5 RapidMiner

RapidMiner [21] is an open-source data mining program which implements many of the different classification-algorithms, including the four which will be used in this thesis. It is both a stand-alone application as well as having a full Java API, meaning that it can be integrated with use in Matlab. It is capable of both training and classifying input data.



(a) (b)

Figure 5.14: The maps generated by DP-SLAM from the model of (a) scene 2, (b) scene 3.

5.6 The following chapters

This chapter has introduced the hardware, software and the other practical aspects of the experiments that have been performed in this thesis. The next chapters will present the actual results of the experiments; they will all be introduced separately at the start of each chapter, but for the sake of overview a short outline is provided here:

- Chapter 6 examines an effort to perform SLAM, i.e. completing a full execution of the chain shown in Figure 2.1, using only the pre-processed distance returns from the D1S sensor and positional data from the rotary encoders.
- Chapter 7 examines how the distances and objects present in the raw data frames from the D1S sensor can be more accurately extracted; i.e. attempt to improve the 'Feature extraction' in Figure 2.1. A mapping using these new distances instead of the distances provided by the D1S sensor is then attempted.
- Chapter 8 examines if the features extracted in Chapter 7 can be classified, and if this new information can improve the mapping between observation and landmark; i.e. attempt to improve the models in Figure 2.1.

6 SLAM with distances

This chapter gives an overview of how distance measurements are used to perform mapping, and examines the out-of-the-box state of the distance measurements from the D1S sensor.

6.1 Mapping with Euclidean distance measurements

This chapter, and the following, describes the most direct attempt at using UWBIR data for SLAM; by using pure distance measurements as exteroceptive sensing. This is realized by for each distance measurement, calculate the Euclidean distances (x_i, y_i) to the sampled distance d_i, relative to the local coordinate system of the robot as shown in Figure 5.2. That is

$$\begin{aligned} x_i &= \cos(\alpha) d_i, \\ y_i &= \sin(\alpha) d_i, \end{aligned} \tag{6.1}$$

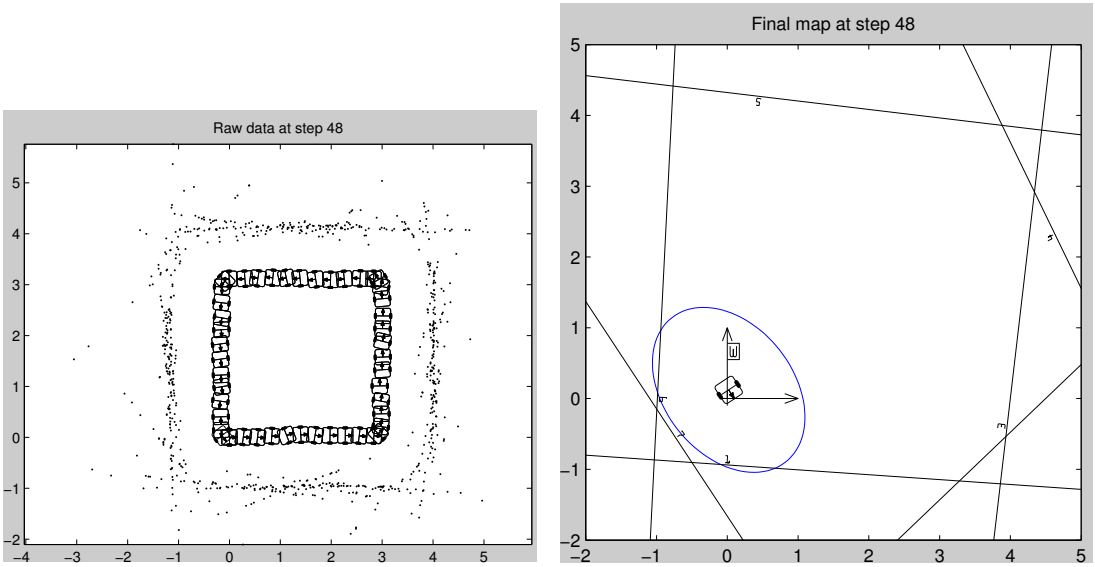
where α is the angle of the rotation by the pan on the robot. These new relative distance coordinates are of the same format most laser range finders use. They can therefore be processed directly by the proposed SLAM algorithms in combination with the positional data from the wheel encoders. The objective is to minimize noise and get minimum-variance estimates of the robot's pose and position, which will produce an estimate of the robot's surroundings.

This chapter will use the processed measurements from the D1S sensor as distances, while Chapter 7 will focus on extracting the distances from the raw radar data.

6.2 Results

The mapping using the distance measurements from the D1S is generated incrementally as the data is processed by the CAS Toolbox, and the result is shown in Figure 6.1a for scene 1. This is the visualization of the surroundings - a map intended for human interpretation. The navigational map which is made up of the edges that are extracted during the execution, as discussed in Section 5.5.3, is shown in Figure 6.1b.

These results show that the robot and its transitional data has been properly set up and collected. Further, the sensor is able to report seemingly correct distances when



(a) (b)

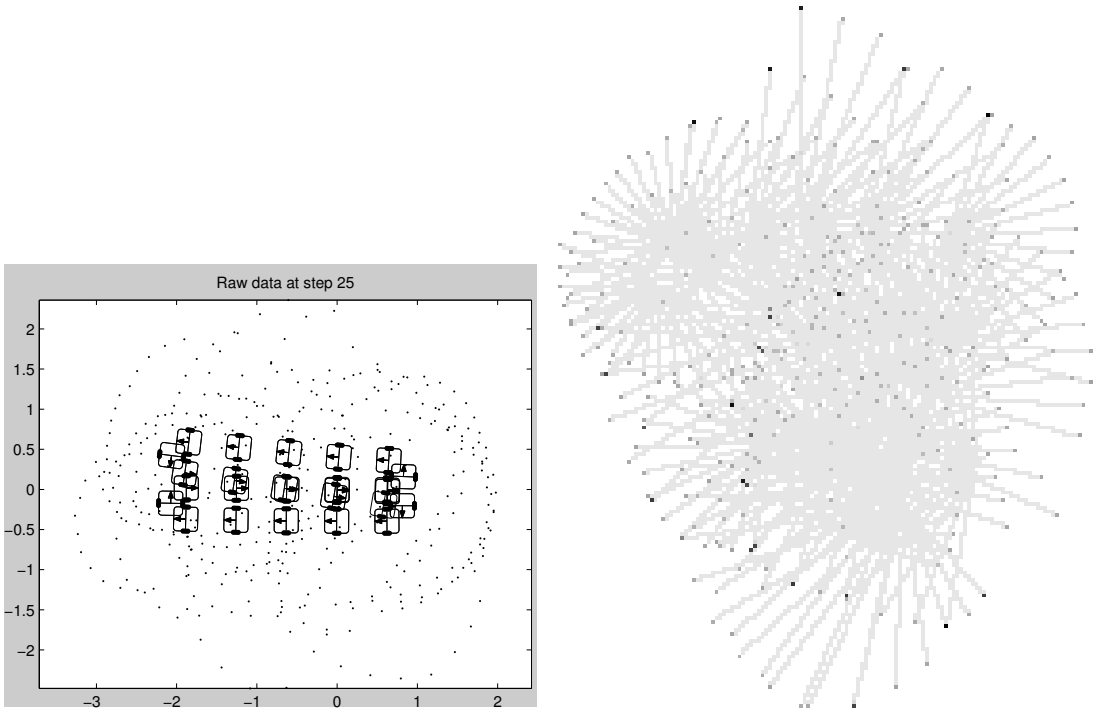
Figure 6.1: The SLAM-results from the CAS toolbox in scene 1: (a) visual map of environment, (b) the extracted lines used as a navigational map.

there are few reflectors and a simple geometry, and it is has been properly calibrated for the scene as discussed in Section 3.2.2.

This setup was included in the thesis to act as a 'calibration-setup' while the software and hardware was implemented, and not for its relevance in SLAM. As an effect of this, the robot followed simpler driving patterns and sampled both propriocetpive and exteroceptive sensors more frequent than is the case in the other two 'real' scenes. Combined with the simplicity of this scene makes it unrealistic to expect this kind of accuracy in the rest of the experiments. Therefore, no efforts to improve these results will be made later in the thesis, as will be performed for the other scenes.

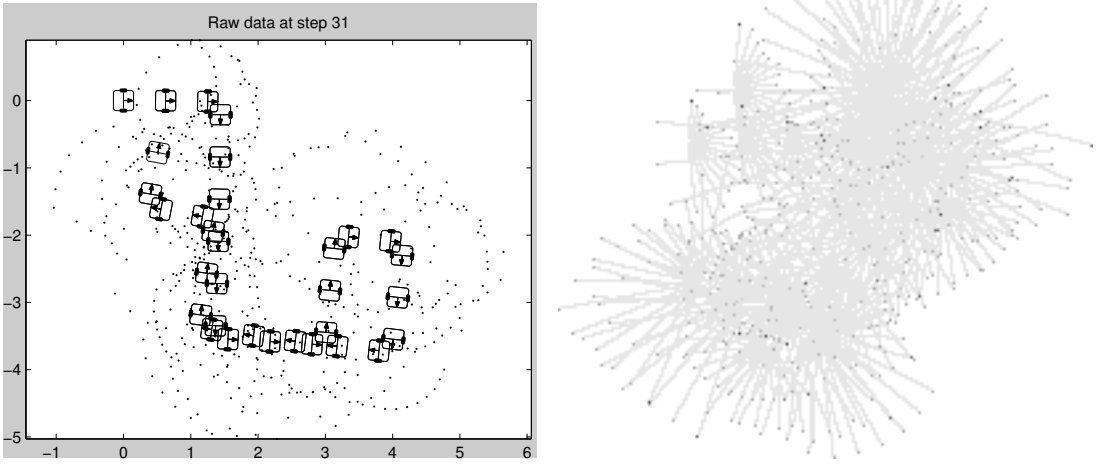
Figures 6.2 and 6.3 show the maps generated by the two SLAM-implementations using the distance measurements from D1S for scene 2 and 3, respectively. In neither scene is the CAS toolbox able to extract any lines, i.e. find any patterns or landmarks in the data. The DP-SLAM algorithm does not extract landmarks in the same way, as discussed in Section 5.5.4, but similarly no sense can be made out of the data.

As can clearly be observed, the out-of-the-box distance measurements from the D1S sensor are not suited to act as direct input for a SLAM-algorithm. This is most likely caused by high levels of clutter due to little or no calibration. This was expected, so the following chapters will therefore investigate whether the reflections contained in the raw radar data transmitted by the sensor simultaneous to the distances can yield any further information.



(a) (b)

Figure 6.2: Map of scene 2 generated by (a) CAS toolbox, (b) DP-SLAM.



(a) (b)

Figure 6.3: Map of scene 3 generated by (a) CAS toolbox, (b) DP-SLAM.

7 SLAM with raw data: Feature extraction

Here we seek to improve the results from the previous chapter by using more accurate distance measurements. As stated in Section 5.3.1, the D1S sensor is not optimized for the kind of sampling being utilized in this thesis, hence the distances it reports holds no guarantee of accuracy. This chapter will therefore examine if the distances can be more accurately extracted from the frames of raw radar data sampled by the sensor, compared to the distance measurements the sensor calculates itself.

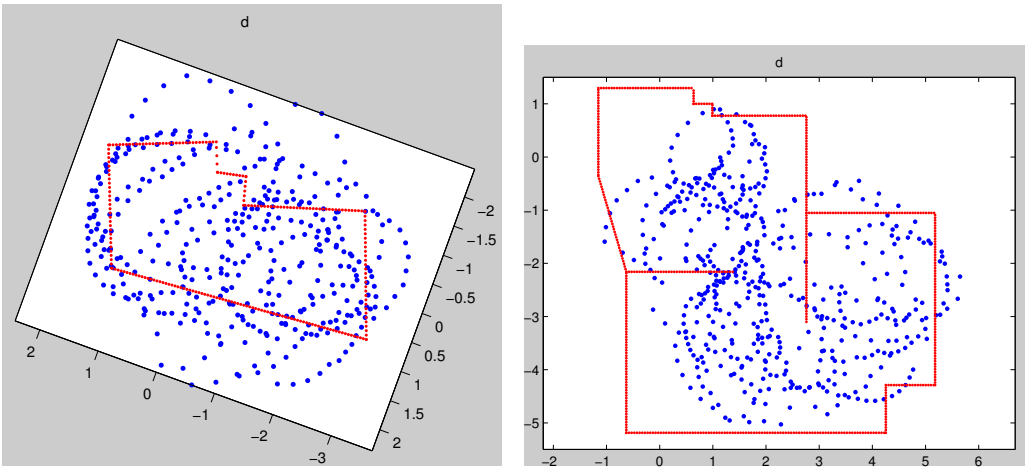
Since the focus of this chapter will be on improving the feature extraction in the SLAM processing chain, only the visualization of the surroundings generated by the CAS-toolbox will be analyzed in the following sections; i.e. only while concluding the results will linear regression and the DP-SLAM algorithm be attempted.

The implementation of the algorithms used in this chapter can be found in appendix A.

7.1 Filtering the raw data

As previously examined, UWBIR raw data is very noisy, and needs to be filtered by removing as much clutter and noise as possible in order to extract the information the frame holds. This information includes both the displacement of the peak(s) representing the actual object(s) in the frame as well as their pulse signatures.

Figure 7.1 shows the distances represented by the highest peaks found in the raw data frames sampled from scenes 2 and 3. They are the same as in Chapter 6, and will be used as reference maps for evaluating the results in this chapter. No filtering has been done, so the distances (blue dots) hardly outline the environment (red lines) at all. This section will first give an overview the filtering algorithms as outlined in Section 3.2 with parameters suited for this thesis, then the results from applying them to the UWBIR raw data is presented.



(a) (b)

Figure 7.1: The distances equivalent to the highest peaks from the raw data, superimposed with a model of the environment: (a) scene 2, (b) scene 3.

7.1.1 Range compensation

The magnitude of the attenuation and dissipation in the D1S sensor can be calculated analytically, but there exist many different models and much theory on estimating these parameters. This, combined with the fact that the effects are believed not to be of great importance for ranges < 4 meters, makes determining these factors fall outside of the scope of this thesis. Instead, a simple experiment was performed to estimate the magnitude of the signal deterioration. A wooden and metal object was moved back and forth in the range of the sensor, and the differences in the signal amplitudes were averaged. This resulted in an approximated range compensation described by the curve shown in Figure 7.2, which is the function

$$\rho = 1 + 0.007i - 0.000003i^2, \tag{7.1}$$

where $i = [1, 2, ..., 1024]$.

7.1.2 Clutter reduction

In this thesis, both SVD and an adaptive clutter map will be implemented.

SVD

As outlined in Section 3.2.1, SVD is a very good clutter-removal scheme, with the downside that it is ill suited for real time operations. Remembering the described stop-

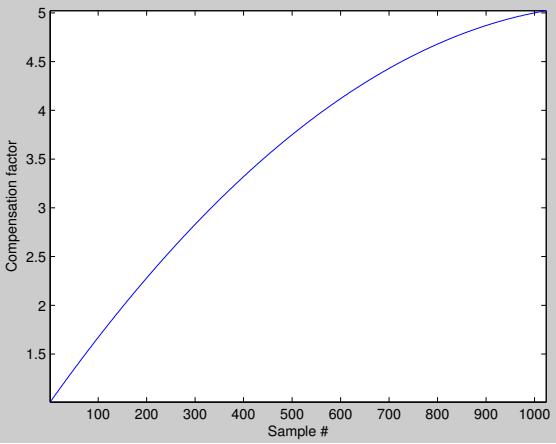


Figure 7.2: Range compensation factor for D1S.

and-shoot sampling scheme of the project, with 16 rotations per stop, we have:

$$Z = \begin{bmatrix} \underline{z}_{1,1} \\ \underline{z}_{1,2} \\ \dots \\ \underline{z}_{1,16} \\ \underline{z}_{2,1} \\ \dots \\ \underline{z}_{n,16} \end{bmatrix}, \tag{7.2}$$

where $\underline{z}_{n,i} = [d_1, d_2, ..., d_{1024}]$ and Z being the set of observations after n stops. It is possible to perform SVD on the collected data between each stop, and thus get incrementally better clutter suppression. However, all raw data must be kept during the execution of the algorithm, as the SVD must be performed on all data each time new samples are added; it can not simply be 'updated' when new data is available. Neither can it factorize too much data due to execution-time limitations, so it must be implemented using a window-function so that it is only applied to the last k samples. It is therefore a cumbersome way of filtering, but if the results from Section 3.2.1 can be achieved in the SLAM algorithm it is worth attempting.

Clutter map

Unlike the SVD, an adaptive clutter map requires no statistical information about the data, and is therefore well suited to real-time applications. The sampling-scheme used in this thesis will result in quite significant changes from frame to frame; i.e. there are few static objects inter-frame. Therefore, a quite small weight will be best suited when put to use in this thesis. This it will preserve most of the information from each new frame, but also suppress most of the static noise present.

7.1.3 Gaussian window

A frequency-domain band filter has been implemented in this thesis. The raw data is first subjected to a Fourier transform (*FT*) to obtain the frequency-domain components of the returned time-domain signal z_i. By averaging many frames sampled by different systems and of different scenes, it is found that most of the energy returned from an actual object is located around sample 180 in the 1024-sample array of frequency-domain components. In order to preserve most of this energy, and decrease contributions from other frequency components, a Gaussian function (bell curve) centered on sample 180 with a standard deviation of 0.3 is used as a window function on the frequency-domain data array. The data is then inversely transformed back to the time-domain.

This algorithm has been developed for use with the D1S sensor, and has shown to be very effective for extracting peaks by smoothing the raw data. However, efforts should be made to prove and optimize its effects on the raw data, which falls outside the scope of this thesis.

7.1.4 Identifying false positives

In the case where there are no objects present in a sample, i.e. when the sensor's field-of-view is 'open-air', the traditional highest peak evaluation will yield a false positive. In order to prevent this, a simple thresholding-function has been implemented, meaning that only peaks with magnitudes higher than a set threshold are identified as objects. Values for the threshold t can range from 5% to 50% max peak-height (mp). A value of e.g. 5% mp means that only the peaks that are above 5% of the height of the highest peak in the collection of raw frames are kept. Typically will $t \approx 20\%mp$ keep about half of the peaks with the D1S sensor.

The max peak-height is not a constant; it will vary from sensor to sensor and environment to environment. In this thesis, it was approximated beforehand by using the strongest reflection found while recording data for the radar data signature database. This raw data will be discussed in Chapter 8.

7.1.5 Results of filtering

Many different parameters for the signal processing algorithms and the combination of these were attempted, and the best are summarized in this section. As a measure for accuracy, the highest peaks extracted after using the different algorithms were compared to the estimated 'ground-truth', sample matrix XY, which are the red lines shown in Figure 7.1. For each extracted peak, the Euclidean distance (equation 6.1) between the peak and the closest point in XY was calculated. This distance is from now on referred to as *the error*. The frequency of the resulting errors are shown in the histograms in figures 7.3 (scene 2) and 7.4 (scene 3). The mean errors are listed in Table 7.1, along with the standard deviations and percentage improvement compared to the raw data.

The arrows between the histograms show the order the processing is performed in. The first histogram shows the errors in the raw data, and the second row shows the errors after the separate clutter removers have been applied. Then, range compensation and Gaussian filtering is performed separately on the SVD-filtered data and these errors are shown, and so on. The y-axis in the histograms represents how many samples are in the bin relative to the total amount of samples from the scene. The y-axis is fixed at the interval 0-35% in the histograms, so the progress can be seen throughout the chain. Similarly, the x-axis represents the values of the bins in meters and shows the 0-1.6m interval. The width of each bin is 0.1m.

Raw data

Figures 7.3a shows the errors for the peaks extracted from the raw, unprocessed data from scene 2. It displays a good 'trend', i.e. the highest bins are those with the smallest errors. However, there is much room for improvement, as discussed in Chapter 6. The errors in scene 3, shown in Figure 7.4a, are much worse; here, the distribution of errors is more uniform over the entire space, and the mean error is as much as 67cm.

Clutter removal

The performance of the clutter removers are shown in figures 7.3b/7.4b (clutter map) and 7.3c/7.4c (SVD), and is according to the figures not very good. Both the clutter map and SVD increased the mean distance error in scene 2 and 3 respectively, whilst the other slightly lowered it. These seemingly poor results are not completely accurate. A visual interpretation of the histograms reveals that both removers filtered many outliers in scene 2, and greatly increased the number of peaks in the 0-0.1m and 0.1-0.2m bins in scene 3. In addition, the general geometry and contours of the environment becomes much clearer when plotting the distances extracted after the clutter removers compared to the raw data. This is shown in Figure 7.5.

The SVD was implemented with a window size of 32 frames; i.e. all frames sampled in the current and previous stop. This way the statistical properties were at least somewhat similar in the 32 frames, making the algorithm able to separate much of the clutter from the signal+noise subspace real-time. The best weight for the clutter map was found to be $w = 0.1$, and with this value the best results were achieved by using both SVD and clutter map. Even with both these algorithms implemented, only the more static sources of clutter, e.g. antenna crosstalk, were removed; more varying sources like ground clutter was still highly present in the frames.

Range compensation and smoothing

The function that had the most effect on the raw data was the range compensation factor, shown in figures 7.3d/7.4d. With this implemented, the ground clutter that appeared in the lowest samples in each frame became lower in amplitude compared to the reflections from the actual objects present. As can be seen from the histograms,

some outliers filtered by the clutter removers became more present again. The mean distance error, however, was lowered 20-35% in both scenes, while also lowering the standard deviations and more than doubling the number of peaks in the best bins.

The Gaussian window did not have the same apparent level of improvement, as shown in figures 7.3e/7.4e. This is because it operates like a smoothing-function, which should not drastically alter the statistical properties of the data. When combined with the range compensation, however, it further increased the visual result in both scenes 2 and 3, while also improving the mean deviations in scene 3. This is shown in figures 7.3f/7.4f.

The thresholding-algorithm was found to yield the most improvement when set to $t = 17\%$ mp. Clearly, this algorithm does not get rid of all false positives, but many of them. Figures 7.3g/7.4g show that the mean error is reduced by another 5% in scene 2, and as much as 25% in scene 3. Many more 'open-air' frames than in scene 2 are most likely the reason for the number of false positives in scene 3.

Complete processing chain

The best combination of processing algorithms for both scenes 2 and 3, as described in detail throughout this section, is summarized in Figure 7.6. It was the same chain that yielded the best result in both scenes, which speaks to the robustness and repeatability of both sensors and algorithms. For scene 2, the mean error between estimated distance and ground-truth was lowered from 34.9cm to 24.9cm - an improvement of almost 30%. Even greater was the effect in scene 3, where the mean error was improved from 67.6cm to 32.1cm - by more than 50%. A thorough discussion of the actual resulting maps follows in Section 7.3.

Table 7.1: Mean distance errors and standard deviations in meters in scenes 2 and 3.

Algorithm(s)	Scene 2			Scene 3		
	Mean	Improv.	Std. dev.	Mean	Improv.	Std. dev.
Raw data	0.349	-	0.285	0.676	-	0.394
Clutter map (C)	0.375	-7.4%	0.26	0.645	4.6%	0.34
SVD (S)	0.338	3.2%	0.26	0.695	-2.8%	0.403
S, range compensation (R)	0.266	23.8%	0.253	0.46	32%	0.379
S, gaussian window (G)	0.351	-0.6%	0.257	0.675	0.1%	0.398
S, R+G	0.266	23.8%	0.237	0.405	40.1%	0.351
S+C, R+G, threshold = 17%	0.249	28.7%	0.247	0.321	52.5%	0.329

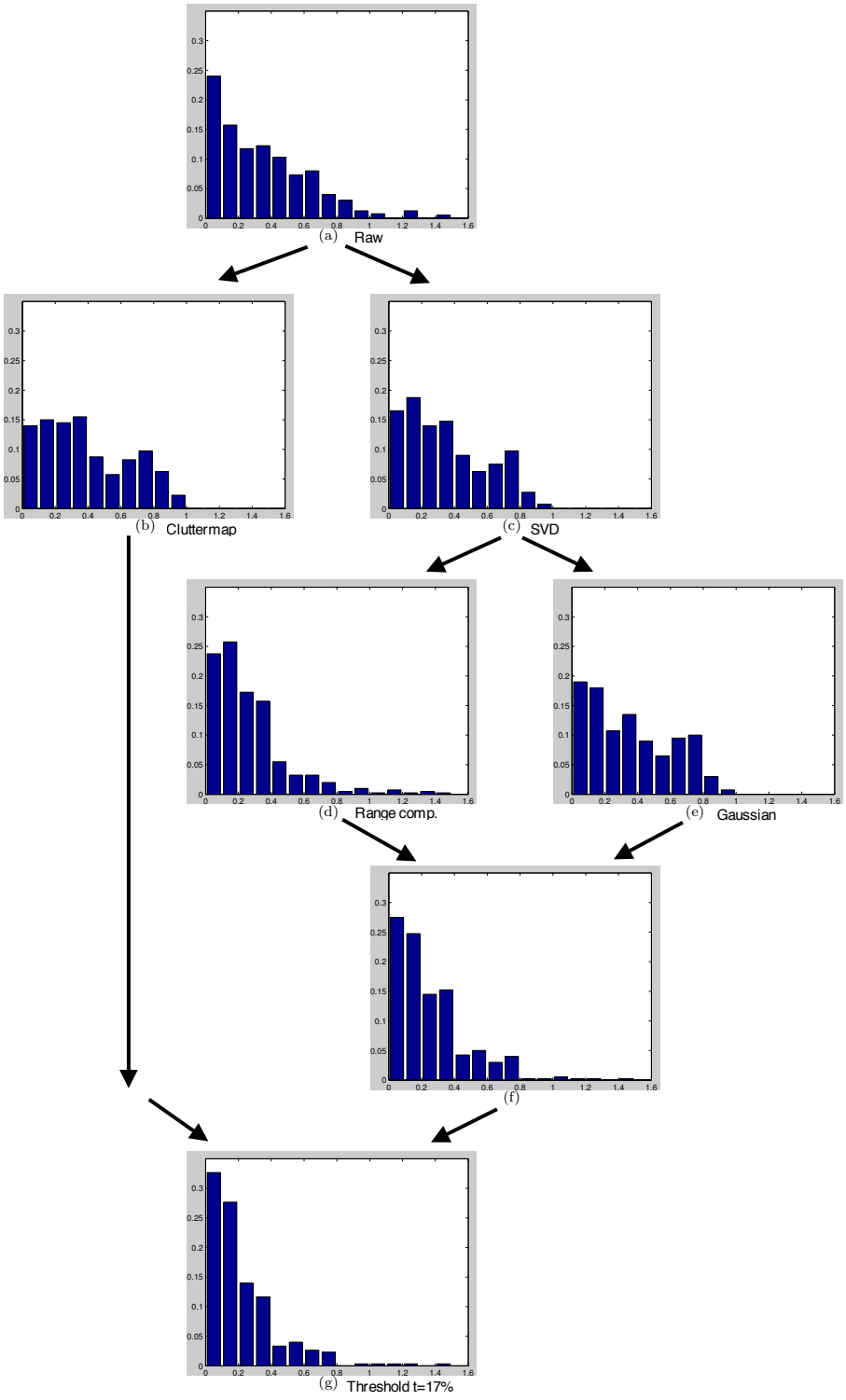


Figure 7.3: All variations in scene 2, see Section 7.1.5; (a) Raw; (b) clutter map [C]; (c) SVD [S]; (d) S, range compensation[R]; (e) S, Gaussian window [G]; (f) S, R+G; (g) S+C, R+G, threshold=17%

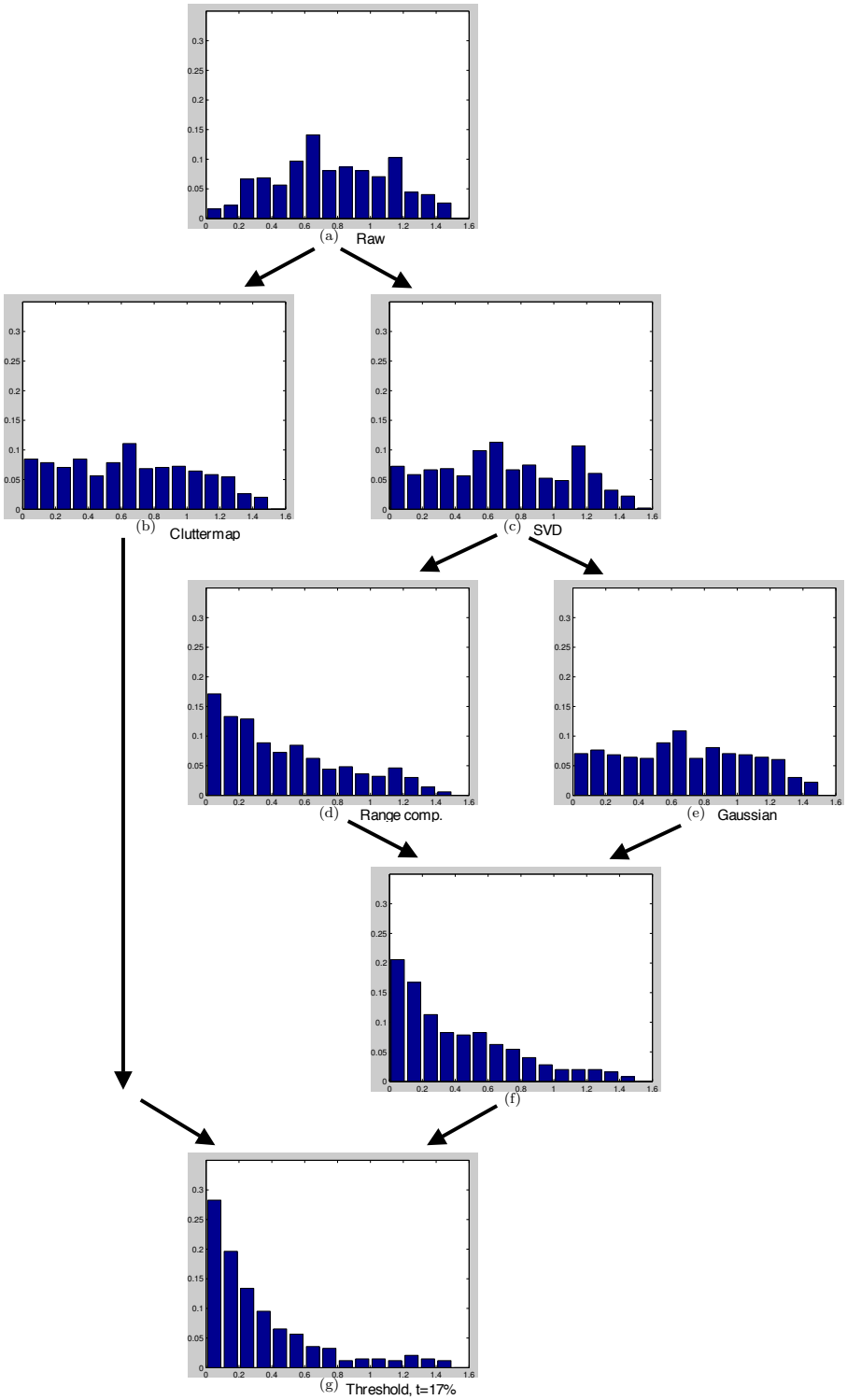


Figure 7.4: All variations in scene 3, see Section 7.1.5; (a) Raw; (b) clutter map [C]; (c) SVD [S]; (d) S, range compensation[R]; (e) S, Gaussian window [G]; (f) S, R+G; (g) S+C, R+G, threshold=17%

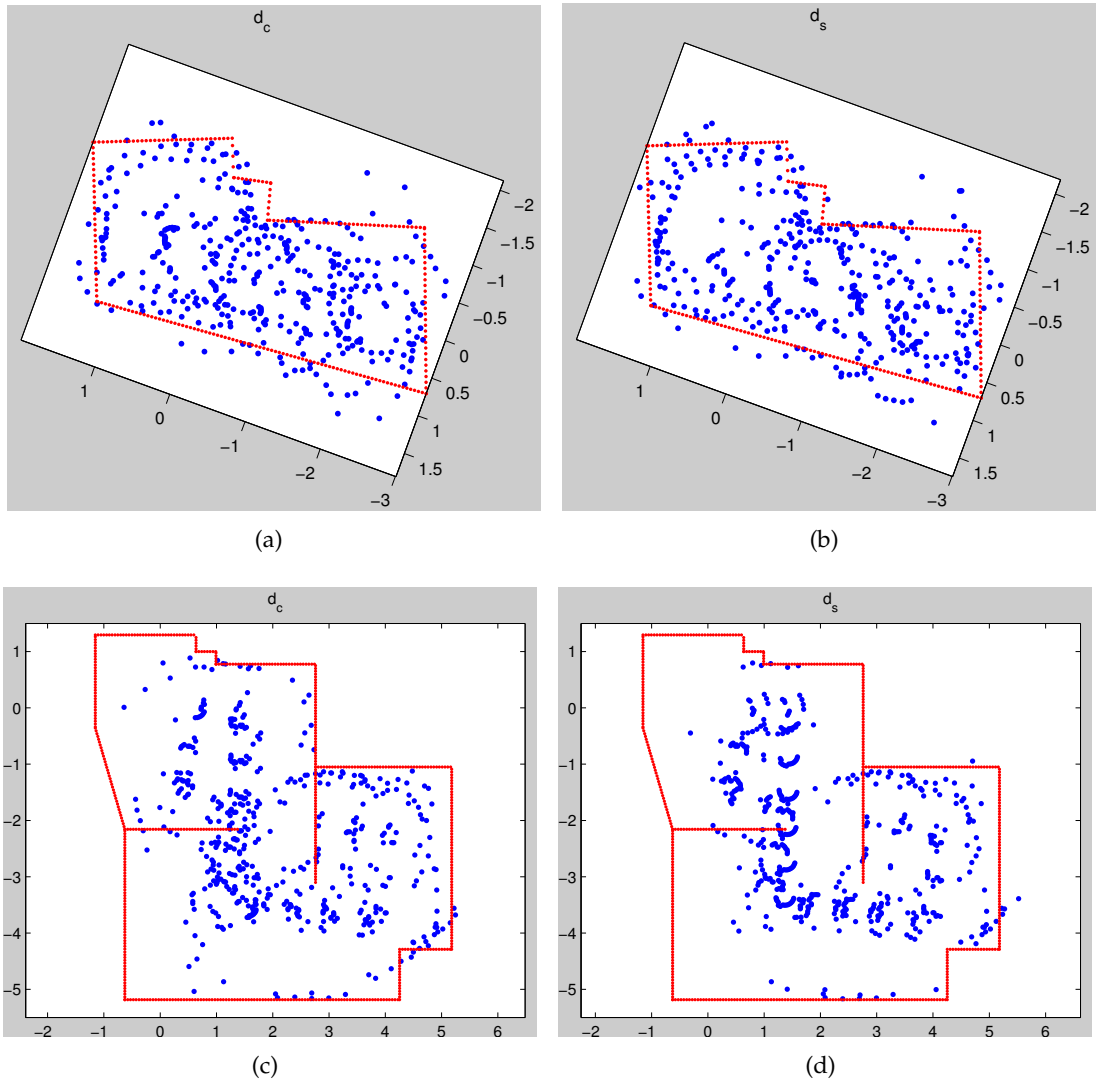


Figure 7.5: The map of scenes 2 and 3 when the raw data has been processed with clutter removers; scene 2: (a) clutter map, (b) SVD; scene 3: (c) clutter map, (d) SVD. The raw data is shown in Figure 7.1.

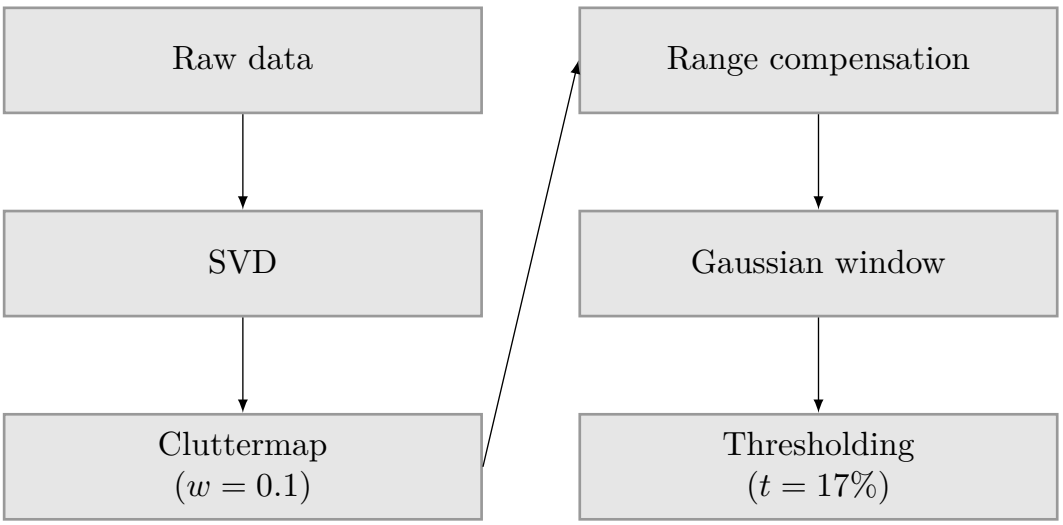


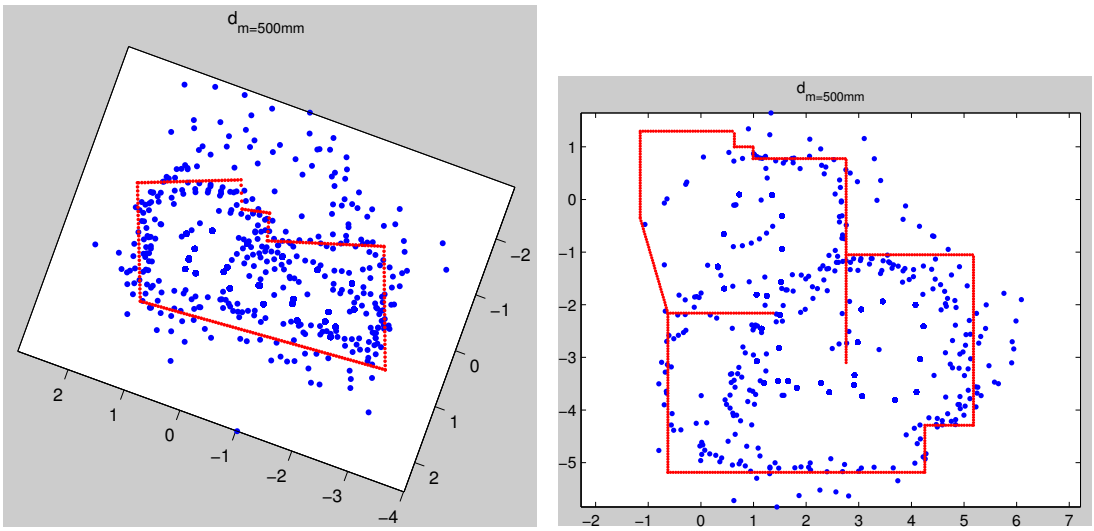
Figure 7.6: The processing chain that yielded the best results.

7.2 Evaluation of multiple peaks

When reporting distances (as opposed to raw data), the D1S is only capable of returning a single distance - the sample with the highest peak in the processed frame. As previously stated, the raw frames from UWBIR systems contain multiple echoes, i.e. reflections from all objects in the scenes. Hence it is possible to extract more information from the raw frames than the sensor is capable of yielding.

Evaluating several peaks in the frame enables the algorithm to operate on more samples than it otherwise would, and will hopefully result in better resolution in the SLAM products. However, it also makes the risk of adding false positives to the pool of distances greater. No special signal processing is needed, other than the methods discussed above.

Evaluation of multiple peaks in scene 2 and 3 is shown in Figure 7.7. Here, only peaks with a minimum separation of 500mm are used. The peaks are chosen from raw data processed with SVD, range compensation, Gaussian filtering and 20%-thresholding. There is not much apparent improvement; there seems to be an increase in the density of the peaks, which is good for the SLAM-algorithms in respect to being able to extract lines and landmarks, but there is also added a number of false positives to the peaks. In the next chapter classification of the peaks will be examined, which potentially can aid in filtering false positives, thus improving the performance of multiple-peak evaluation.



(a) (b)

Figure 7.7: The map of scenes 2 and 3 when multiple-peak evaluation is added to the processing chain. The raw data is shown in Figure 7.1.

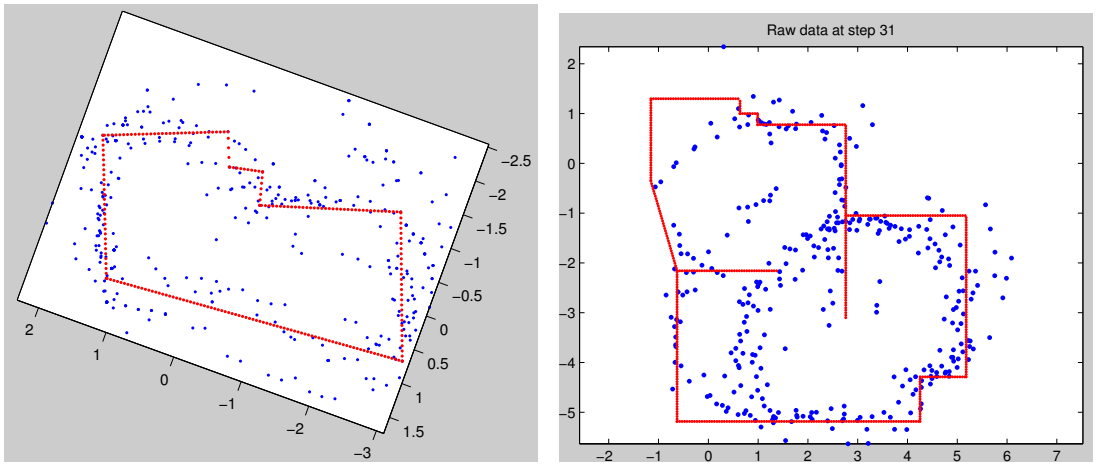
7.3 Using filtered raw data in SLAM algorithms

The best visual maps, as produced by the processing chain in Figure 7.6, are shown in Figure 7.8. Although the resolution is poor and erroneous measurements are present, the general geometry and layout of the environment can be recognized. Compared to the maps shown in Figure 7.1, the filtering has greatly improved the mapping abilities of the robot. As can be observed in Table 7.1, the mean distance errors have sunk by 28.7% and 52.5% for scene 2 and 3 respectively.

Figure 7.9 shows the results from using the processed data for actual SLAM in the CAS toolbox. This refers to the process of combining the individual measurements to extract lines from the data ('Filter' and 'Feature extraction' in Figure 2.1). The CAS toolbox is able to extract a few lines in each scene, compared to none in Chapter 6, although few of them represent actual walls. Highlighted in Figure 7.9 are the bounding data points the toolbox selected for creating some of the lines. As can be observed, they do not represent measurements of the same surfaces, or even a surface at all. Therefore, the next chapter will focus on using classification algorithms for improving the selection of data points used for extracting lines.

The maps the DP-SLAM algorithm was able to create are shown in figures 7.10a and c for scene 2 and 3 respectively. As can be observed, there were too few data points for the algorithm to yield any readable result. Therefore, figures 7.10b and d show the same data, only up-sampled by a factor 10. I.e. between each 16° pan step of the sensor, 10 new 'measurements' have been made by interpolating between the actual samples. This is not a very accurate approach, but accurate enough for yielding a visible map

displaying the trends in the data. For scene 2, the resulting map is very little coherent with the map shown in Figure 5.14a, which is the map generated from the ground-truth model. The map from scene 3 is slightly better and the geometry of the scene can be guessed at. This can be because the samples in scene 3 (Figure 7.8) contains fewer extreme outliers than scene 2, so the interpolation therefore has greater effect. All in all, DP-SLAM is more dependent on accurate and quantities of data compared to the CAS toolbox, since the CAS toolbox works by extracting lines from arbitrarily few data points.

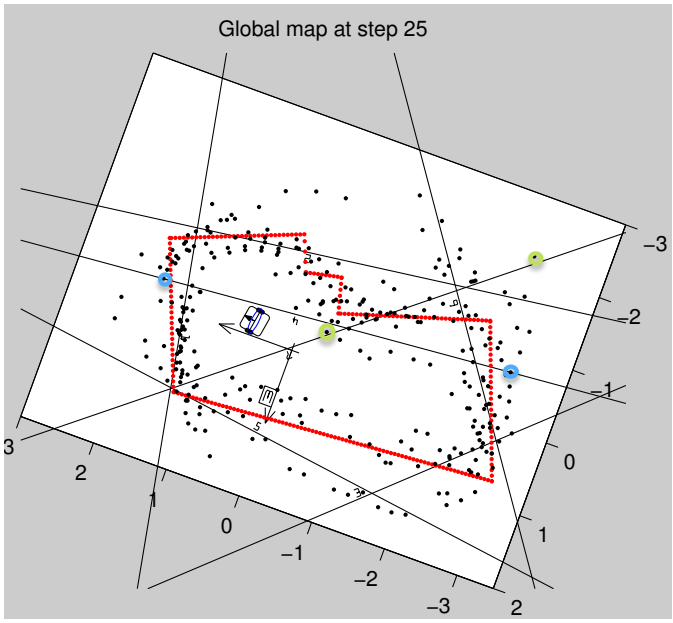


(a) (b)

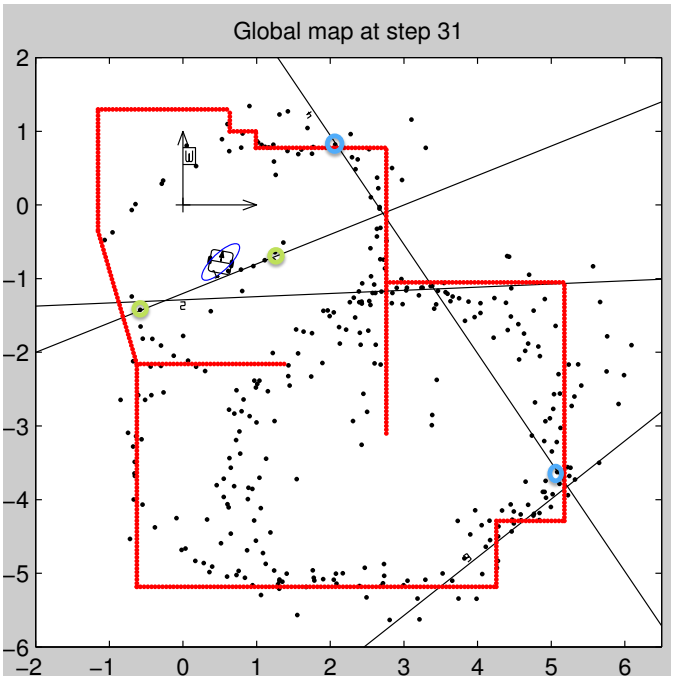
Figure 7.8: The best 'visual' maps superimposed with the actual surroundings: (a) scene 2, (b) scene 3. The raw data is shown in Figure 7.1.

7.4 Results overview

All in all, the maps that have been generated in this chapter are great improvements from the ones in Chapter 6, but it is clear that the D1S sensor does not match traditional LIDAR-based sensors in resolution in non-occluded environments. The SLAM algorithms are not able to generate navigational maps from the data thus far. Therefore, the next chapter will examine feature classification for UWBIR data with respect to improving the SLAM-performances.



(a)



(b)

Figure 7.9: The extracted lines by the CAS toolbox from the best measurements in (a) scene 2, (b) scene 3.

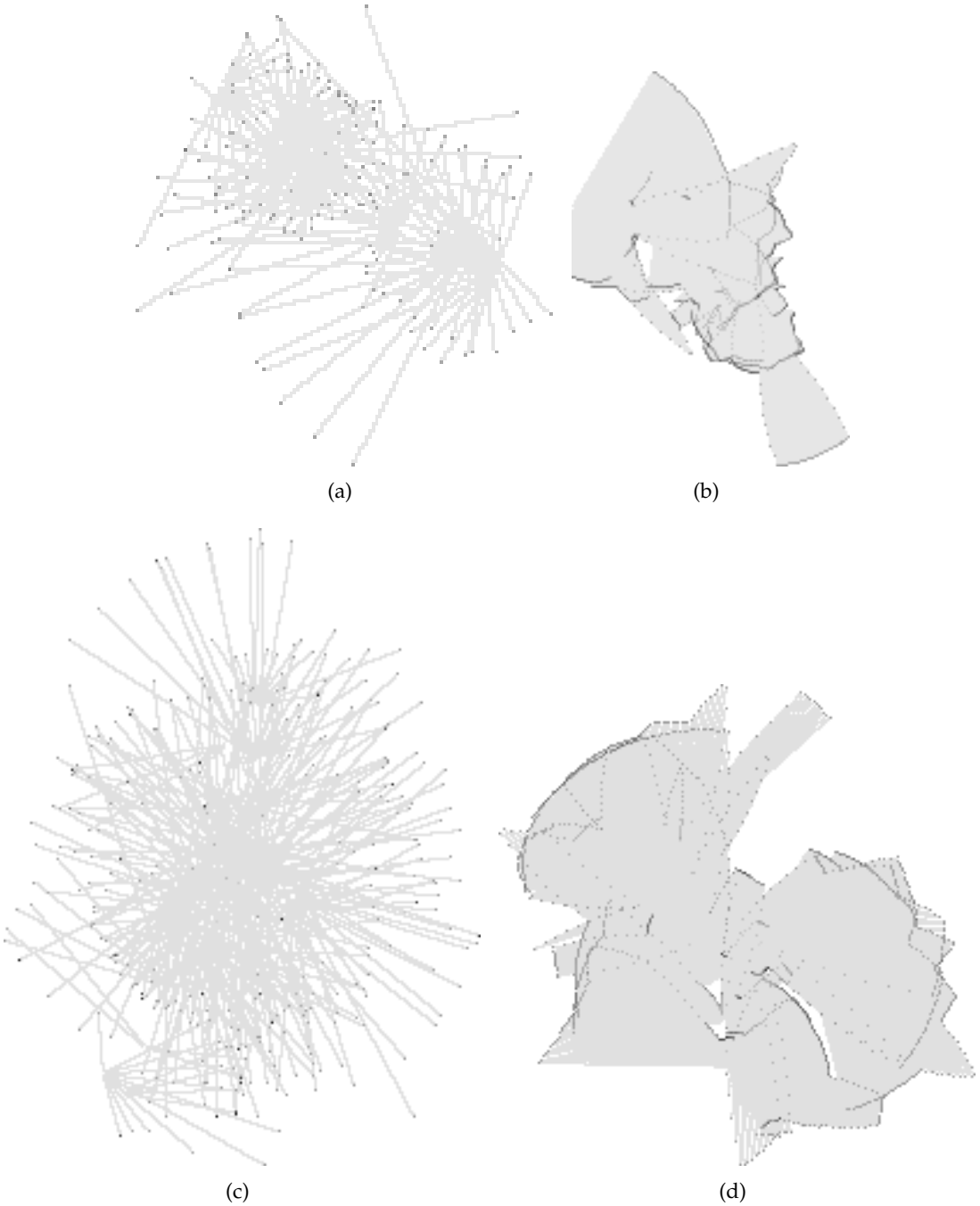


Figure 7.10: The maps generated by DP-SLAM from scene 2: (a) distances, (b) interpolated distances; scene 3: (c) distances, (d) interpolated distances.

7.5 Improving the visual results

If a map intended for human interpretation is the primary interest, there are ways of improving the results presented so far in this chapter.

7.5.1 Synthetic aperture radar processing

One such method is called *synthetic aperture radar* (SAR) processing [1][7][25]. When sampling the environment, the robot clearly sees the same scenes more than once. The idea behind SAR is to gain higher resolution, especially in azimuth for radar data, as well as signal-to-noise ratio (*SNR*) by synthesizing a long(er) aperture. This is realized by coherently combining the different samples of the same resolution cells in the scene. A resolution cell is a quantized area of the real-world scene, e.g. a 10 cm$\times$10 cm pixel. This can be achieved in several ways. One method is *back projection* (also known as Kirchhoff migration and delay-and-sum), which is to summarize the contributions from all the samples that have seen the same scene. In pseudo-code, the algorithm is simply:

```
for i in x-resolution cells
    for j in y-resolution cells
        for k in measurements
            A(i,j) += k(i,j);
        end
    end
end
```

Here, $A(\cdot)$ represents the amplitude in the point, or pixel, in the map.

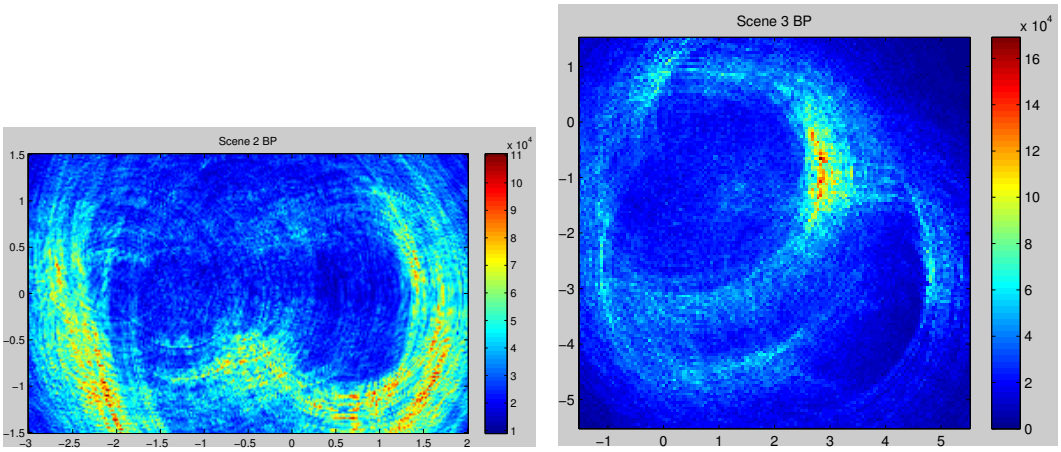
This adds up to higher amplitudes where there are objects present without increasing the level of the noise since we assume uncorrelated white noise that will be cancelled out, hence increasing the SNR. This methodology requires very accurate positioning data, down to the fraction of a wavelength, in order to perform as well as it potentially can. It is also a very time-consuming algorithm, potentially ill suited for real-time operations depending on the desired resolution. It is therefore outside of the scope of this thesis and most SLAM purposes to implement to the fullest, but a simple effort has been made.

The radar data used for the back projection was subjected to the processing chain shown in 7.6, with the exception of the thresholding function. This was omitted in order to have as much data to iterate over as possible. The results are shown in Figure 7.11. In scene 2, the walls are clearly highlighted, with the exception of the wooden wall at the top. Additionally, the walls seem to appear in 'layers', hinting at that the positional data might not have been accurate enough to successfully close the loops that occurs when the robot makes several round trips in the environment.

Scene 3 shows that one of the walls reflected very well, and most of the other surfaces are also identifiable. The area in the bottom left that contained several items in the scene appear cluttered, as expected, and all in all only small amounts of noise and

clutter can be observed. Compared to scene 2, this can be caused by that the robot only made one round in the environment, as well as that the surfaces were spaced farther apart.

The back projection algorithm performed well, but made it apparent that the positional data could be more accurate and a greater quantity of ranging data would be desirable.



(a) (b)

Figure 7.11: The results of back projection SAR processing: (a) scene 2, (b) scene 3.

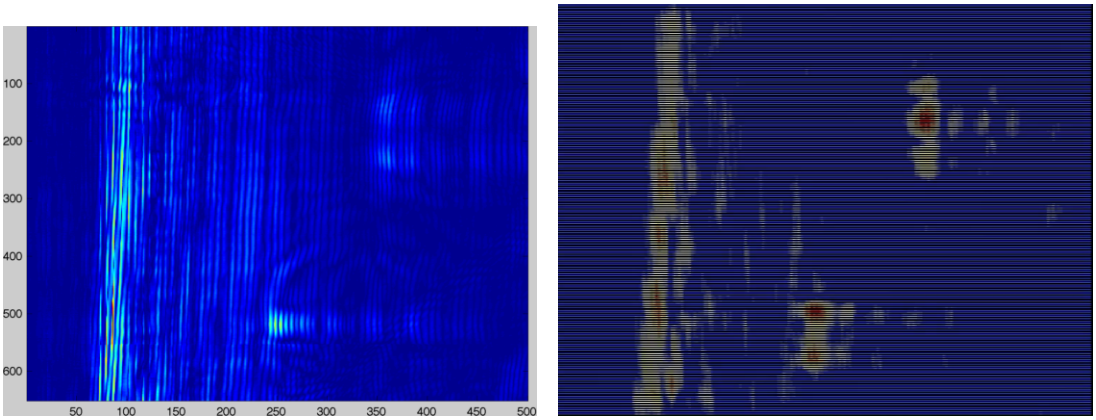
Through-wall imaging

As discussed in Section 3.1.2, UWBIR sensors can be used to see through objects due to the lower parts of the frequency spectrum of the signals. This can be a very attractive feature for tactical mapping, if not for navigation, and has therefore been attempted in a small scale in this thesis.

The experimental setup was for the robot to drive along a brick wall with two metal objects placed behind it, as shown in Figure 7.12. The setup allowed for a much higher level of accuracy in positional data than were the case in the main experiments performed in this thesis, as the sampled area was much smaller and the robot drove only straight forward. Additionally, data was sampled every 5cm, resulting in much more data to iterate over.

The raw data recorded by the robot was first filtered with SVD clutter-removal and range compensation, resulting in Figure 7.13a. It was then subjected to back projection, resulting in Figure 7.13b. As can be observed, the back projection is very effective, clearly highlighting the objects that are actually present in the scene. The accurate positional data and frequent sampling made the algorithm perform significantly better than was observed in the previous section, although the execution time for this small scene was about the same as for the much larger scene 2 above.

Figure 7.12: Photo of the sampled scene. The wall is 30cm higher than the robot.



(a) (b)

Figure 7.13: Scene after being processed with (a) SVD and range compensation, and (b) back projection.

8 SLAM with raw data: Feature classification

In this chapter we will look at how object recognition in UWB radar systems can be used to enhance a SLAM algorithm further. With the results from the previous chapter, the extracted features will undergo classification, and attempts will be made to feed back this information to the SLAM algorithms. This can potentially be helpful in extracting landmarks, hence improving the performance of the algorithms.

8.1 Implementing and training the classifiers

As reviewed in Section 5.5.5, the RapidMiner software will be used in this thesis. All the proposed classifying algorithms are therefore already implemented, so this section will examine the training of the classifiers.

8.1.1 Example frame

Figure 8.1 shows one full frame from D1S, after the raw data has been processed with SVD. The frame is a sample of the scene shown in Figure 8.2, used for early experiments in this thesis. As can be seen, all three objects in the scene (radiator, person and drywall) are present in the sample, and their individual signatures are shown in Figure 8.3. At a glance they all appear quite similar in shape, but the analyses done in [23] shows that they are all unique enough to classify the type of object from the reflection.

For this thesis, we will assume that there is only one reflection in the signal; in other words, we will only attempt to identify one object in each frame. Mainly this is of conceptual reasons: when the robot maps its surroundings, it is logical to only indicate a positive where there is something obstructing the path of the robot, and not analyze what may be located further ahead. Searching through the 1024 data points in each frame for potential secondary or even tertiary objects would also greatly increase the computational effort of the algorithm; when classifying only one object we just analyze the 'area' around the highest peak in the signal, which is what was found in Chapter 7. With a pre-trained classifier, this operation is low-cost enough to be a part of a real-time execution.

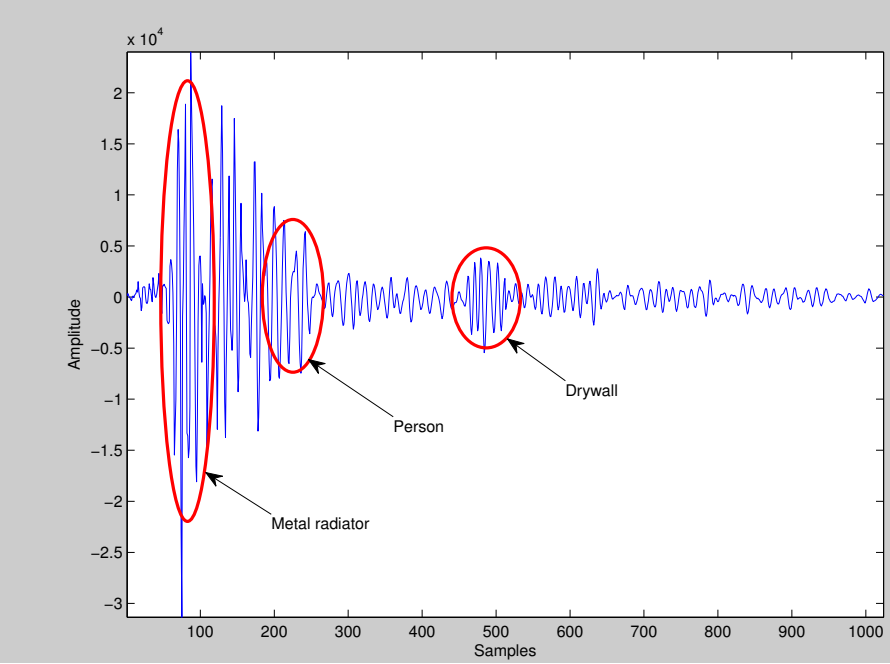


Figure 8.1: Radar data from D1S with three objects present.

8.1.2 Feature extraction

Assuming that the features represented by the extracted points as discussed in length in Chapter 7 are correct, all that remains in order to classify the objects is to extract the reflection from the object from the rest of the frame. The width of the pulse signature within the frame is not fixed. In an anechoic chamber, with only a perfect point scatterer in the range of the sensor, the reflection from the object will always be identical ([12]). In the real world this is not possible, because of the multitude of scatterers in every scene. Factors such as the angle of incidence, spacing to other (large) reflectors and the spatial extent of the object itself all impact how the transmitted pulse deforms when reflected. In this thesis, where the number of classification categories and the complexity of the environment are kept to a minimum, 60 samples are chosen to represent the object. These are determined by the sample with the highest peak in the processed radar data, and its 59 closest neighbors. This covers the spatial extent of

$$\frac{4000\text{mm}}{1024\text{samples/mm}} \cdot 60\text{samples} \approx 234\text{mm}, \tag{8.1}$$

and should be enough to get all parts of the (main) reflection.

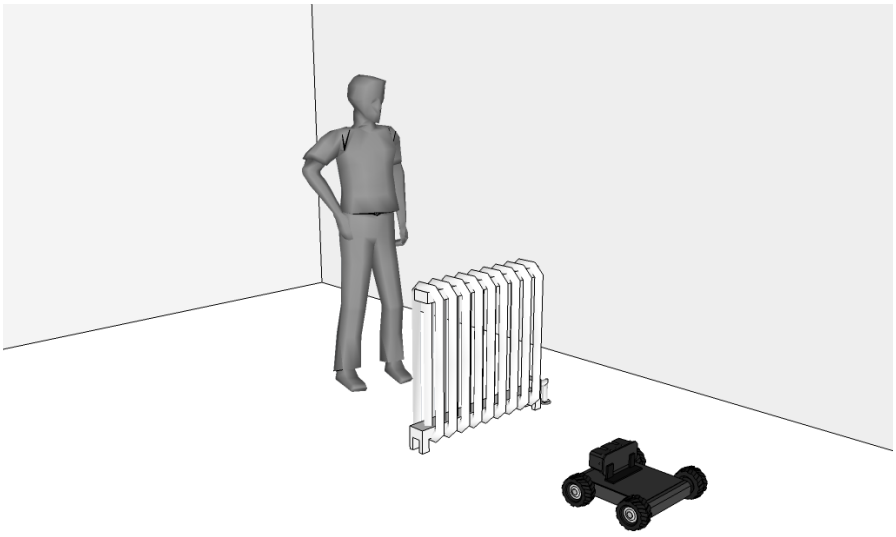


Figure 8.2: Illustration of sampled scene.

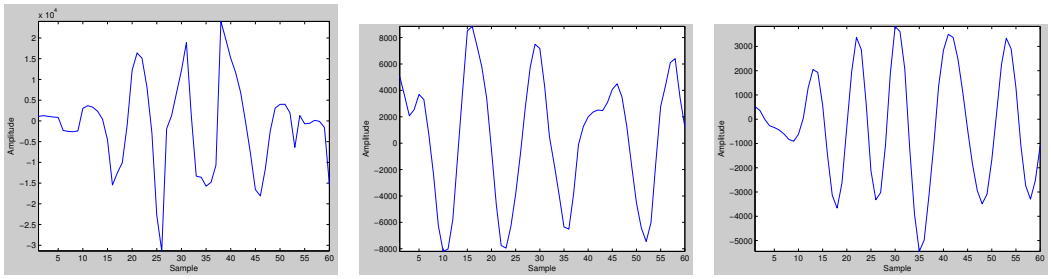
8.1.3 Signature database

In order to do object recognition through pulse signature matching, the algorithm needs a database with 'known' signatures. The purpose of the database depends on the classifier; some algorithms use it to construct and train neural networks, while others compare the sample to be classified to the entries in the database to estimate its class.

In all cases, the database should contain as many and as accurate pulse signatures as possible. The database used in this thesis consists of 200 samples of each of these five classes:

- Metal
- Wood
- Drywall
- Brick wall
- Person

They were selected because they are all commonly found in indoor environments, and are different enough to be relatively easily separated through their signatures. 200 samples of each class were chosen in order to keep the database of a manageable size, whilst still containing a fair amount of each category. The data for each class was acquired through sampling the objects listed in Table 8.1 at different distances (ranging from 10cm to 3m) and angles (all inclinations in the 40° field-of-view). The sampling



(a) (b) (c)

Figure 8.3: Radar data frame from D1S, focused on (a) radiator, (b) person, (c) drywall.

was done by the same D1S sensor as was later used in the experiments in this thesis. This thesis mainly investigates SLAM for scenes where little or no a priori information is available. Therefore the objects were not the same that were present in the scenes from the main experiments, only of the same material, in order to keep the generality of the problem. The raw data from these samples was then processed with SVD to remove clutter. Finally, a 60-entry subarray of the processed data centered on the highest peak reflected from the current object was isolated and normalized before adding it to the database.

Table 8.1: Objects used for acquiring pulse signatures.

Class no.	Class	Sampled objects
1	Metal	Metallic lockers and drawer
2	Drywall	Three different drywalls
3	Person	Two different persons
4	Wood	Plywood plate and wooden Table
5	Brick wall	Two different brick walls

8.1.4 Supervised learning

The supervised classifiers were tested by using a leave-some-out scheme, where the signature database consisting of 1,200 samples was split into three equally sized parts. The algorithms were trained on two of the sets, and the remaining was used for testing. Then the test-set was used as a training-set, and one of the training-sets became the test-set, and so on. Finally the results from the test rounds were averaged to obtain the total accuracy, which are presented in Table 8.2.

The kNN-algorithm requires only one parameter: k, i.e. how many neighbors each node in the input should be compared with in the database. The best results were here obtained with $k = 9$. For the MLP and SVM, the default parameters in the RapidMiner-implementations were used; further work should include optimizing these parameters

for UWBIR data, which has not been investigated in this thesis.

Table 8.2: Average class prediction on the test sets by using machine learners.

Class	kNN	MLP	SVM
Metal	61.19%	80.30%	81.82%
Drywall	98.51%	100.00%	98.51%
Wood	77.61%	85.07%	80.60%
Brick wall	100.00%	100.00%	100.00%
Person	90.91%	78.79%	80.30%
Total	85.63%	89.03%	88.24%
Elapsed time	61ms	24ms	21ms

We see from the results that kNN performs quite well during the training. The execution time, which was believed to be a big hindrance for the kNN to be usable, is absolutely acceptable because of the small size of the database. Increase this however, and the kNN will quickly become slow.

Overall, all the results from the training are better than expected. The algorithm with the seemingly best performance is the MLP. For almost all cases, theory states that the SVM should outperform alternative algorithms - this, combined with the fact that the training data does not have as high diversity as it maybe should have, suggests that the MLP might have started to move towards overfitting. If this is the case, it should become apparent when classifying the real data.

Most surprising is that all three algorithms successfully classified the brick wall signatures 100% of the times, as well as nearly 100% on the drywall. 'Person' also has a higher success rate than wood and metal. The surprising part is that all of these 'materials' are compounds, and was believed to have more varying signatures than wood or metal. Drywall in particular, classified with 100% certainty by two out of three algorithms while varying greatly in both thickness and percentage of each of the compounded materials it is made out of. One explanation could be that both wood and especially metal have a higher reflectivity than the compounded materials, and hence small geometrical variations affects the deformation of the pulse more than with e.g. a drywall. Another feasible explanation is that parts of the reflection from the wooden and metallic objects can have been modified, or even removed, by the signal processing used to isolate the reference pulses, to a greater degree than with the compounded classes. This could be because much of the clutter in the samples comes from reflections from other wooden and metallic objects in the scene, such as the floor and wires in the walls. Sampling the reference pulses in an anechoic chamber and keep the signal processing to a minimum can test this.

8.1.5 Unsupervised learning

The main problem with using k-means as a tool in SLAM is that there is no analytical way to determine how many different objects, and hence clusters, there are in the scene. The graph shown in Figure 8.4 is the result obtained from categorizing the database entries into 5 clusters - the same amount as there are classes in the database - using the kMeans-function in Matlab. These show that the pulse signatures indeed are separable without using a supervised learning scheme, but in an actual SLAM-setting where the number of different reflectors is unknown, matching these results will be impossible. What we can do, however, is use these clusters as a database, and then adapt them when new data is obtained during the SLAM algorithm. Again, we have a problem with assigning the number of clusters, as there are doubtlessly more than these 5 classes in the scene. One way of solving this is to set a threshold for each cluster; if the new data is located too far away from any clusters in the input space, assign it its own cluster and remap all the clusters with this new information.

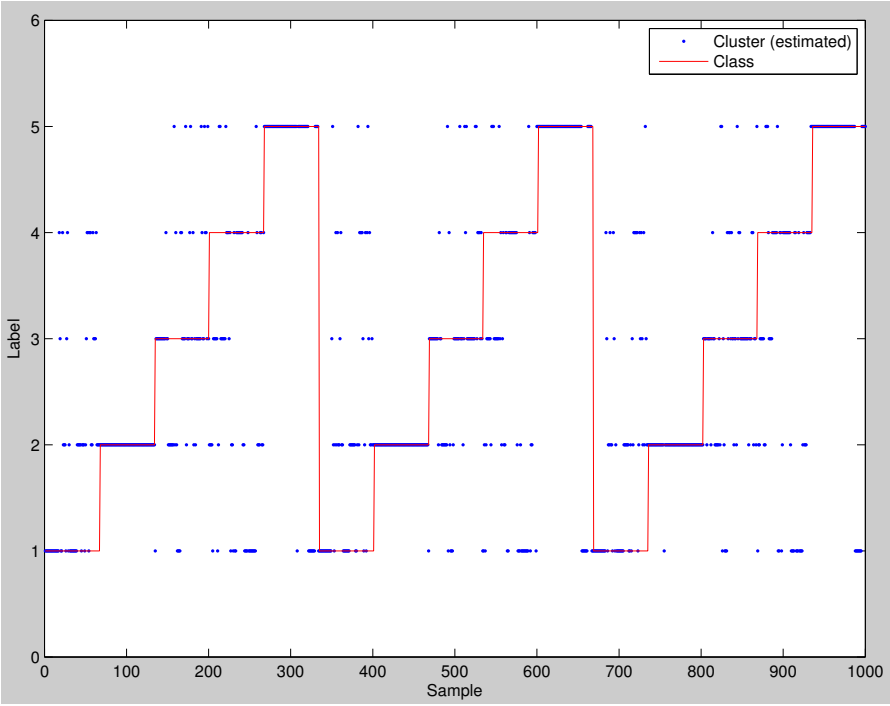


Figure 8.4: Estimated clusters from the database with k=5.

8.2 Classifying the project data

Using the trained algorithms reviewed in the previous sections, this section presents the results from classifying the data recorded in the main experiments performed in this thesis.

8.2.1 Supervised learning

The parts of the raw data from the scenes subjected for classification was extracted from the 'best' distance measurements found in the previous chapter, similar to how the data for the database was obtained as described in Section 8.1.2. Estimating how well the algorithms were able to classify these parts is not in itself a very accurate exercise, since the ground-truth models as shown in figures 5.10 and 5.11 contains many deviations and do not present accurate solutions themselves as discussed in Section 5.4. Therefore, the results shown below will not contain 'absolute' accuracy.

Figures 8.5a,c,e show all the distance measurements from scene 2 with their classes as estimated by the kNN-, MLP- and SVM-algorithms respectively. The measurements that were correctly classified are shown in figures 8.5b,d,f. 'Correctly' is here defined as the measurement being located 20cm or less away from a node in the model of the environment (shown in Figures 5.10 and 5.11) with the same class. Figure 8.6 shows the same for scene 3.

In tables 8.3 (scene 2) and 8.4 (scene 3), the accuracy and precision of the classifications have been attempted quantized. Here, *accuracy* is defined as how many of a class present in the scene was correctly classified. For example, there were 27 correctly classified metal-measurements using kNN in scene 2, out of 109 total metal-measurements in the model, giving an accuracy of 24.8%. *Precision* is defined as how many of a class were correctly classified. Using the same example, there were 66 measurements that were classified as metal, of which 27 were correct, resulting in a precision of 40.9%. The last row in the tables is the mean of both accuracy and precision, and acts as a measure of the combined performance of the classifier.

Since the focus of this chapter is to improve the performance of the mapping in its simplest form, i.e. simply the geometry and layout of the room, the 'human'-class from the training epoch was omitted from the actual scenes.

The overall results are disappointing. As with the training data, the drywall and brick classes yielded the best performance for all classifiers, especially where accuracy is concerned, which suggests that the classification is at least partly successful. All classifiers also showed remarkably similar performances, with the SVM being only marginally better in both scenes. All in all, though, the processes seem to produce 'random' results. This will be discussed further when summing up the classification in Section 8.2.4. An indication of whether the data is indeed as random as it might seem here is whether an unsupervised learning algorithm is able to cluster the measurements in a sensible way or not, i.e. successfully group measurements of the same material.

Table 8.3: Class prediction on SLAM features in scene 2 by using supervised learners. The best and worst results are highlighted in blue and red respectively.

Class	kNN		MLP		SVM	
	Accuracy	Precision	Accuracy	Precision	Accuracy	Precision
Metal	24.8%	40.9%	28.4%	36.5%	26.6%	41%
Drywall	50.5%	35%	50.5%	39.7%	55.8%	43.8%
Wood	12.7%	19.2%	8.9%	16.7%	11.4%	26.5%
Brick wall	47.1%	17.8%	41.2%	13.5%	35.3%	7.9%
Overall	31%		29.4%		31.1%	

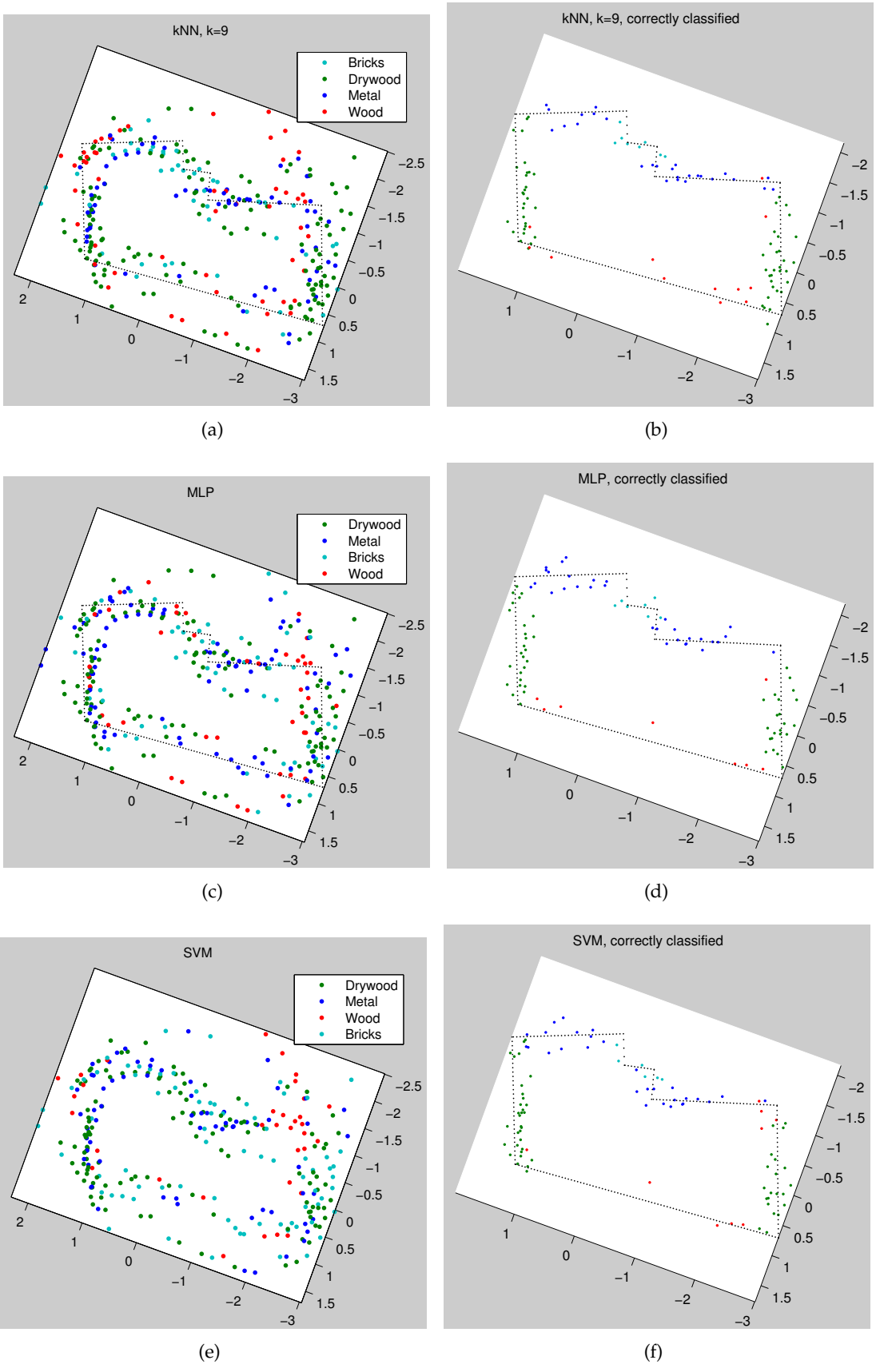
Table 8.4: Class prediction on SLAM features in scene 3 by using supervised learners. The best and worst results are highlighted in blue and red respectively.

Class	kNN		MLP		SVM	
	Accuracy	Precision	Accuracy	Precision	Accuracy	Precision
Metal	15.7%	13.6%	17.7%	13.2%	15.7%	15.4%
Drywall	30.3%	37.4%	31.7%	31.9%	28.9%	32.5%
Wood	18.7%	16.3%	22.7%	18.7%	17.6%	19.5%
Brick wall	17.3%	15.5%	15.5%	15.5%	23.1%	15.8%
Overall	20.6%		20.9%		21.1%	

8.2.2 Unsupervised learning

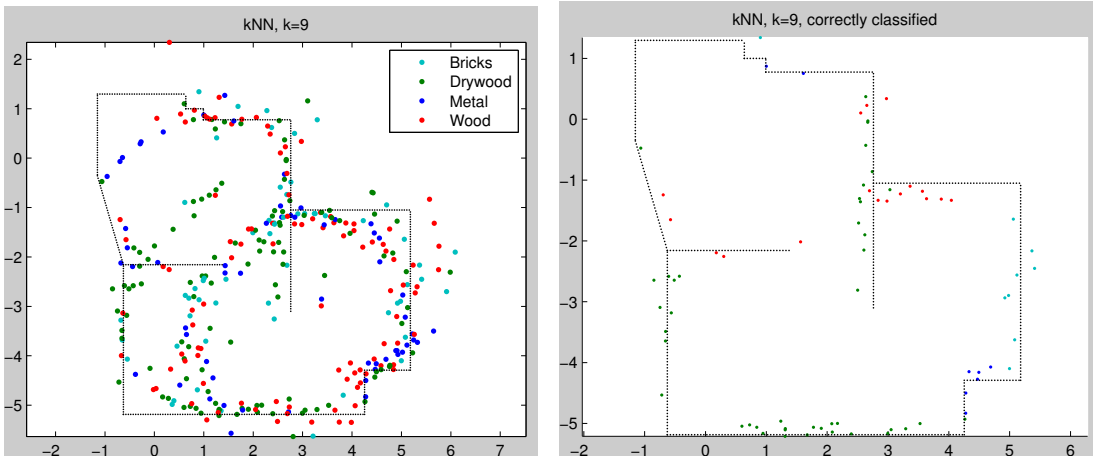
The results from kMeans-clustering with different values for k are shown in Figure 8.7.

Some reasonable clusters are found, e.g. top left corner in scene 2 and in fact an entire cluster which seems to hold most metal measurements in scene 3 with $k = 4$. Again, though, the data seems very random. Judging from the results from the training epoch, as reviewed in Section 8.1.4, the reflection from the different materials should be unique enough to be able to cluster more informatively than what was accomplished here. This might imply that the extracted data points from the last chapter as summarized in Section 7.3 was not accurate enough, i.e. many false positives were extracted. This will be further discussed in the next section.



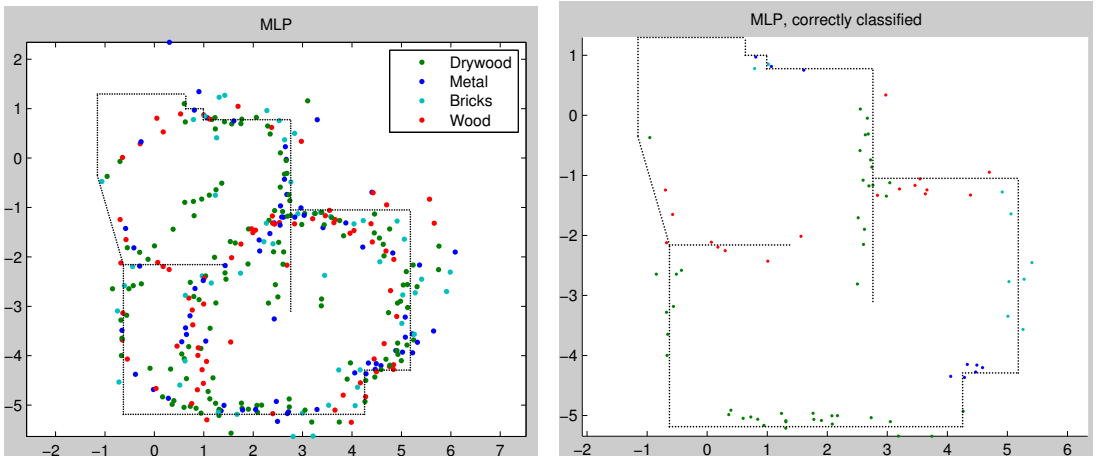
(a) (b) (c) (d) (e) (f)

Figure 8.5: Results from classifying the extracted features in scene 2 with different algorithms: kNN, k=9: (a) all features, (b) correct features; MLP: (c) all features, (d) correct features; SVM: (e) all features, (f) correct features.



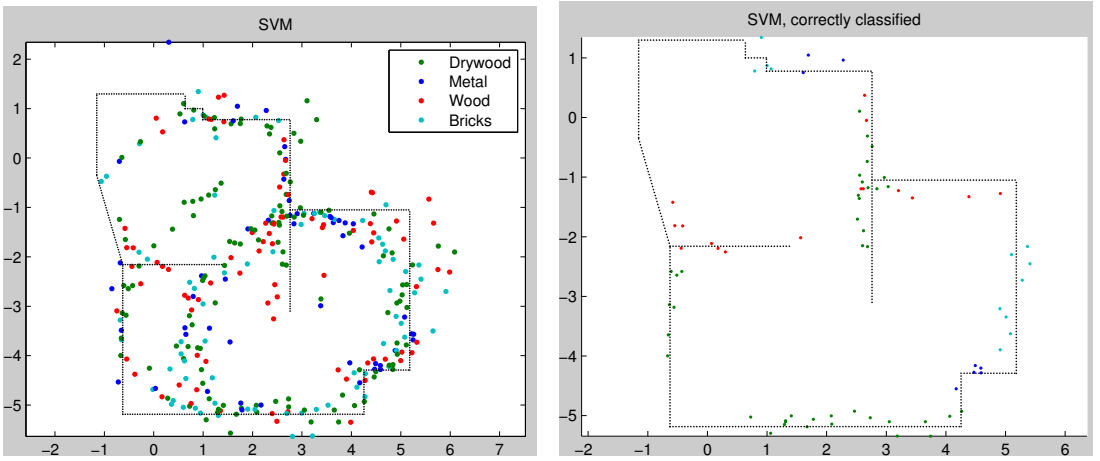
(a)

(b)



(c)

(d)



(e)

(f)

Figure 8.6: Results from classifying the extracted features in scene 3 with different algorithms: kNN, k=9: (a) all features, (b) correct features; MLP: (c) all features, (d) correct features; SVM: (e) all features, (f) correct features.

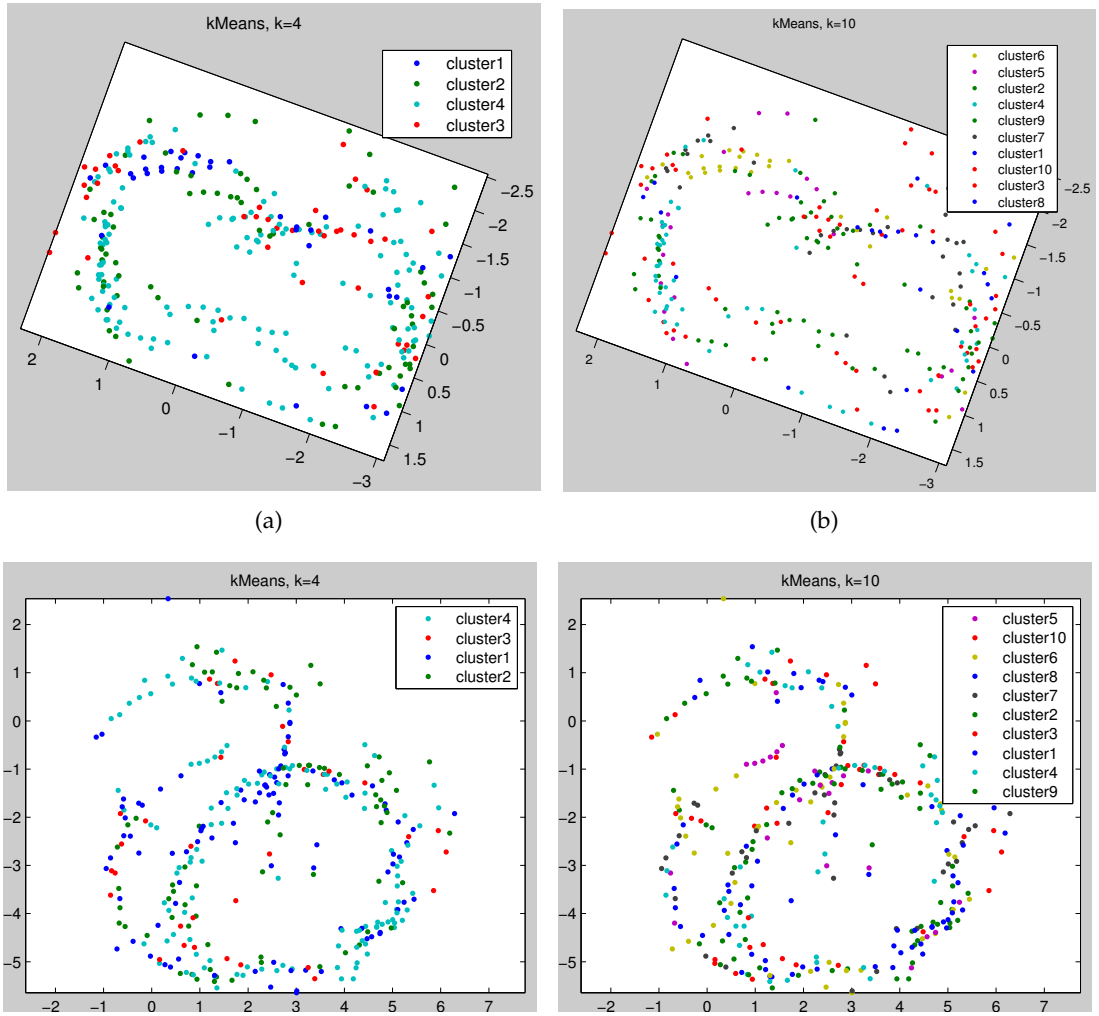


Figure 8.7: Results from clustering the extracted features using kMeans, scene 2: (a) k=4, (b) k=10; scene 3: (c) k=4, (d) k=10.

8.2.3 Identifying false positives

Using the confidences of the predictions from the kNN and MLP algorithms, an attempt was made at filtering out false positives from the data points extracted in the previous chapter. That is, if a prediction had a low confidence (<0.9), it was assumed to be a false positive. This, however, yielded no significant results, and as often as not seemingly 'correct' predictions were reported to have lower confidences than known false positives. This will be further discussed when summarizing the classification results.

8.2.4 Classification summary

The results summarized in Section 8.2 were not as uplifting as the ones from the training epoch as summarized in Section 8.1.4. Scene 3 in particular displays very little coherence with the actual surroundings. This is due to a number of reasons; firstly, the data sampled in the different scenes most likely contain much noisier data than the samples used in the database. This again is caused by several contributions: the database samples were obtained in a much more controlled setting, where the exact distance to the objects being sampled was, if not known, then at least much easier to obtain from the raw data than was the case in the actual scenes because of fewer and relatively distinguishable objects. Where as in the experimental scenes, the sensor often had several surfaces or objects in its field-of-view, resulting in radar returns containing more noise in the form of echoes from several strong-reflecting sources. In addition, the data used for the signal database was sampled in a more continuous fashion than the experiment data was. The SVD was then performed once on the entire dataset, instead of incrementally as it was in the main experiments. This results in a much more effective SVD-filtering, potentially making the optimal pulses in the database have different shapes than the pulses extracted in the previous chapter had.

Secondly, the database contained reflections from only a few objects that were assumed to be representative for their general classes. This assumption may not have been correct, making the results from the training epoch appear artificially good. The training epoch results may even have moved towards overfitting, due to too little variation in the data.

Thirdly, the features extracted in the previous chapter may not have been an actual object at all, so the data subjected to classification wasn't represented in the database. As stated in Section 5.4, the surfaces in the scenes were rarely comprised of a single material, but rather a mix of several, whereas the samples in the database were all homogenous. In addition, as described in Section 8.1.2, the part of the data frame estimated to hold the pulse deformed by a potential object covers a spatial extent of over 20cm. Hence, there is also a danger of there being multiple objects in the pulse subjected to classification, as well as one non-represented object or no object at all.

Nevertheless, as can be observed from figures 8.5b,d,f in particular, some objects have indeed been classified 'correctly', and the outline of the environment can be guessed at even with only the correctly classified data. Therefore, the rest of this

chapter will examine how this information can be used to enhance the SLAM method as summarized in Section 7.3.

8.3 Implementing in the SLAM algorithms

As discussed in Section 4.1.1, successfully identifying corners greatly aids any SLAM algorithm. However, the poor azimuth-resolution of the sensor makes identifying corners based on signal strength hard, and it is therefore omitted.

As discussed in Section 5.5.3, the CAS toolbox uses linear regression to extract line segments from the data points and combine these to create full lines, hence producing a navigational map for the robot. In [3] line extraction is split into two subproblems:

- Segmentation: *which* data points contribute to the model (line)
- Fitting: *how* these data points contribute

The *how* of it is performed by linear regression, and Figure 5.13 (the lines extracted from the models of the scenes) illustrates that the toolbox does this well. Using the results from the classification, the focus will therefore be on improving the segmentation, i.e. *which* data points should contribute to a line.

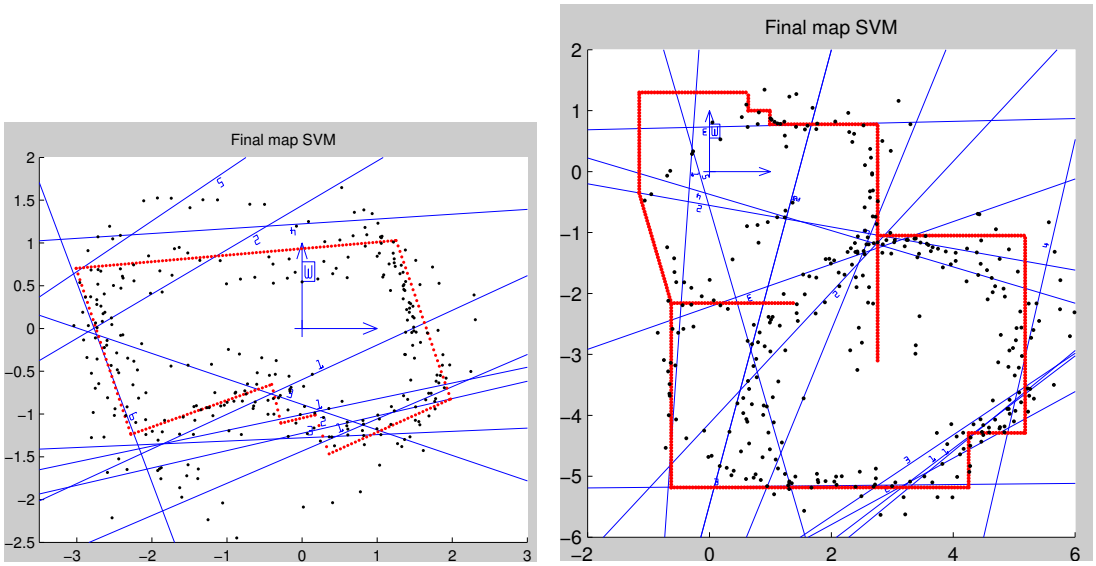
8.3.1 Increased quality of data points

There are already mechanisms for performing the segmentation of the data points in place as described in [3]. Using the classification results can further enhance these. Using the notation from the section on linear regression earlier in this thesis (4.1.4), the measurements $\underline{x}_m, \underline{y}_m$ are split into groups of known classes, $\underline{x}_m^c, \underline{y}_m^c$. These are more likely to contain points in a line without outliers, as they most likely are measurements of the same surface. Separate regression analyses are performed on these c clusters of data points instead, i.e. equation 4.2 becomes

$$\underline{y}^c = \alpha^c \underline{x}_m^c + \beta^c. \tag{8.2}$$

In the CAS toolbox, this is simply achieved by running c iterations of the algorithm that extracts the lines, one for each set of classified data, and attempt to match all these segments with previously found segments. This resulted in the maps shown in Figure 8.8. The data used was the SVM-classified measurements, as shown in figures 8.5e and 8.6e, since the SVM performed best of the classifiers as discussed in the previous section.

As can be observed, this technique improves the results for the CAS toolbox SLAM approach using line-based feature extraction, compared to the results from the previous chapter shown in Figure 7.9. Many lines are extracted, of which most form non-existent lines based on false positives or erroneous combinations of data points. But several also form completely or partially correct features, and given even more accurate data - both distance measurements and number of correctly classified measurements - this technique shows great promise.



(a) (b)

Figure 8.8: The lines the CAS toolbox were able to extract from the SVM-classified data from (a) scene 2, (b) scene 3. The red lines are the model of the scene, the black dots are the data points used for line extraction, and the blue lines are the extracted lines.

8.3.2 Increased quantity of data points

While the CAS toolbox operates by extracting lines from arbitrarily few data points, DP-SLAM simply draws densities depending on the measurements and pose of the robot. Therefore, it depends more on having a higher quantity of data points than the CAS toolbox does. Obtaining more data points for the DP-SLAM algorithm can be achieved by interpolating between the existing measurements that are available, as discussed in Section 7.3. Using linear regression to extract lines as described in the previous section, an arbitrary amount of data points can be added to the measurements. Motivated by the results shown in Figure 5.14, which shows the maps the algorithm produced from the model of the scenes, this can potentially improve the performance of the DP-SLAM algorithm. On the other hand, adding a linear regressor to DP-SLAM is in some ways conceptually wrong, as one of the selling points of the algorithm is that it needs no feature extraction to perform well.

8.4 SLAM performance

Many algorithms and approaches have been discussed in this thesis, and conclusions have been made throughout. All in all, how well have they assisted the mobile robot in performing SLAM using an UWBIR distance sensor?

Using pure distance measurements from the D1S sensor showed that the processing

performed online by the sensor is not enough to yield correct measurements in a complex scene. When subjected to more extensive signal processing as reviewed in Section 7.1.5, up to 50% more accurate measurements were extracted from the raw radar data. From these measurements the general geometry of the scenes could be seen, but they were only accurate enough for the CAS toolbox to be able to extract lines with much deviation from the real map. This is illustrated in figure 7.9. It is possible that other signal processing algorithms would have performed even better in filtering the data, but drastic improvements are unlikely. The main reason for the lacking results throughout the thesis is most likely too inaccurate positional data. SLAM algorithms can handle noise and some level of inaccuracies and use the ranging data to adjust for this. But if the positional data is too coarse in resolution, the mapping can often appear skewed. This was also observed when using the backprojection imaging algorithm, where the surfaces appeared in layers when the robot made several passes of the same places.

The machine learning algorithms also suggested that the measurements were somewhat inaccurate. The results from classifying the experiment data were significantly poorer than from the test data. Still, this new information enabled the toolbox to extract several lines from the measurements. Far from all of the lines were represented in the actual scene, but caused by outliers and falsely classified measurements. With more accurate distances, the performance of the classifiers and hence linear regressors are expected to increase greatly.

Potential ways of improving the accuracy of the extracted distances will be presented in Chapter 9.

The algorithms - and combinations of these - developed in this thesis shows that UWBIR can successfully be used in SLAM. Although there is a general lack of accuracy, many of the methods can be further improved, and all have been at least partially successful.

Part IV

Conclusion

9 Conclusion

This chapter summarizes the discussions and results presented in this thesis, and recapitulates the most interesting conclusions drawn. Lastly, suggestions for future work and topics for further investigation are proposed.

9.1 Summary

Chapter 1 gave a brief introduction to the field of robot navigation, and presented the problem statement for this thesis and the motivation behind it: can UWBIR-based sensors contribute to solving the SLAM problem?

The SLAM problem was defined in chapter 2, and two different approaches to solving it, EKF and particle filtering, were presented. It was stated that data association and computational complexity and the trade-off between these are amongst the main challenges that still remain fairly open. Lastly, example SLAM products were shown.

Chapters 3 and 4 presented a closer look at the UWBIR technology, both in theory and practice. Means of filtering the raw data consisting of noisy reflections from the objects present in the scene being sampled were summarized. A discussion then followed as to how these objects can be *extracted* from the filtered data and *classified* by their pulse signature. The pulse signature is given by the deformation the transmitted EM impulse is subjected to when reflecting of the object, and is unique enough to be recognized by machine learning algorithms. This can be used to enhance the performance of SLAM algorithms using UWBIR-based sensor measurements. These factors formed the basis for the experiments that were conducted. The processes involved in implementing the proposed experiments were a very big part of the thesis, and the resulting setups were described in detail in chapter 5. The platform for the experiments was a 4WD rover controlled remotely over WiFi and equipped with a D1S UWBIR distance sensor. The D1S served as the sole exteroceptive sensor, and rotary encoders attached to the wheels were implemented for proprioceptive sensing. The rover was programmed to drive in three controlled indoor environments and record both raw radar data and distance measurements calculated by the sensor itself. In chapter 6, the calculated distances were used as the exteroceptive sensor data for the CAS toolbox. When these were combined with the data from the wheel encoders the validity of the setups were confirmed; i.e. the resulting maps seemed accurate in robot pose and position, but lacking in resolution.

Based on these early results, chapter 7 sought to improve the resolution of the maps by extracting more accurate distance measurements from the raw data than the sensor itself was able to provide. Several signal processing algorithms were attempted, and the filter chain which resulted in the highest level of accuracy included both SVD and an adaptive clutter map for clutter suppression, as well as range compensation, bandpass-filtering and (normalized) amplitude thresholding. This yielded an improvement from the distances extracted from raw data of more than 50% at best. Still, the SLAM maps were very lacking in resolution and usability compared to e.g. LIDAR-based sensors. Even more effort could have been put into developing better feature extraction algorithms, but it was still concluded that the radar data cannot compete with these types of sensors where pure resolution is concerned.

Chapter 8 therefore focused on feature classification with UWBIR data, and investigated whether this can be used to improve the SLAM method. The classifiers that were tested were kNN, MLP and SVM. The results from the training epochs were very promising, with averages surpassing 90% accuracy. The data sets from the training epoch were therefore used as a database of pulse signatures for classifying the data from the main experiments. This resulted in less positive and partly inconclusive results, with classification accuracies ranging from 20-30%. Several possible causes for this were discussed, with the most likely being inaccurate distance measurements and complex, compounded materials compared to the test data. Even though the accuracy of the test data could not be matched, the classified experiment data improved the performance of the SLAM algorithms, and is believed to be able to do so even further with more accurate data.

In conclusion, this thesis has provided an implementation of a mobile robot utilizing an UWBIR sensor for performing 2D, indoor SLAM. Signal processing and classification algorithms for enabling a successful mapping have been found, although resulting in maps of variable quality. It is therefore concluded that UWBIR systems should not be used as stand-alone sensors for exteroceptive sensing in SLAM. However, they still possess several attractive features that should be examined further. These are outlined in the following section.

9.2 Future work

Here are listed some suggestions for topics that would be interesting to look further into, based on the experiences gained in this thesis.

Sensor fusion.
An implementation where the works of this thesis are fused with more traditional LIDAR or camera-based approaches can potentially result in SLAM products with greater resolution as well as the robustness and classification-abilities of the UWBIR technology.

SAR processing.
Imaging radar realized through SAR is a big field of research, and several papers

are published for UWBIR systems as well. For use in autonomous navigation, however, few works exist.

Signature-based feature extraction.
In this thesis, the highest peak was found in the current frame of processed raw data, and was chosen as the distance measurement. This process can be replaced by performing a regression analysis between known pulse signatures and the frame, and only report a distance if an object is found. This is a more time-consuming operation, but it should eliminate the open-air problem and other erroneous measurements.

Regressed classification.
Similar to the previous point, classification was used in this thesis to improve the linear regression. An alternate approach, given more data, is to reverse this by using regression to improve classification (through more accurate distance measurements.)

Outdoor and 3D SLAM.
As previously stated, there is little to no conceptual difference for using the UWBIR sensor outdoors, and it is perhaps in these environments it potentially excels compared to optical sensors. 3D-SLAM is more challenging, but there is ongoing research dedicated to creating arrays of UWBIR sensors, hence making interferometry and thus 3D-SLAM more easily available.

Gaussian window algorithm.
The frequency-domain bandpass-filter used in the thesis has proven to be very effective, but it requires work in the form of proof and parameter optimization.

Machine learning parameters.
The parameters used for the machine learners in this thesis were chosen by trial-and-error, as no efforts have previously been made to find optimal values for UWBIR data.

Real-time implementation.
The algorithms used in this thesis were all implemented to be real-time friendly, but are not ported and standardized for any robot. A full implementation where a robot uses these algorithms for navigation is therefore of great interest.

Greater quantity of data.
Repeating the experiments conducted in this thesis with the robot collecting more data than was the case here (both shorter distances between each stop and more samples per stop) can also prove to have a great impact on resolution.

Appendix A: Matlab scripts

Listed below are the signal processing algorithms used in this thesis, as well as some helper Matlab functions. For detailed description, see comments in the code.

getpeaks.m
The function used for extracting the highest peaks from raw UWBIR data. Takes a variable number of input arguments specifying the signal processing algorithms the raw data should be processed with.

crosstalkrem.m
SVD clutter-removal algorithm.

cluttermap.m
Clutter map algorithm.

rangecompensation.m
Function for adding the empirical range compensation factor to UWBIR data.

fft_gaussian.m
Bandpass-filtering algorithm, which filters the frequency spectrum data with a Gaussian distribution.

sar.m
Backprojection algorithm for filtered data and predefined map size.

gui.m
The simple GUI that was used for communicating with the robot.

Listing 1: getpeaks.m

```
1 function [ peaks ] = getpeaks( rawdata, varargin )
2 % The signal-processing algorithm, can be called with the following
3 % arguments:
4 % q = reshape to matrix
5 % r = rangecompensation
```

```
% z = nulling, followed by how many
% c = cluttermap, followed by weight
% s = svd filtering
% d = convert sample# to distance
% t = threshold, followed by value
% f = gaussian window
% m = multiple peak, followed by separation in mm

[m n] = size(rawdata);
peaks = zeros(m,1);
procdata = rawdata;
peaksep = 40;

threshold = 0;  % Threshold for peak-height
matr = 0;       % Matrix-form (25x16 instead of 400x1)
dist = 0;       % Convert index to distance
multi = 0;      % Multiple-peak processing

i=1;
while i < nargin
    switch varargin{i}
        case 'r'
            procdata = rangecompensation(procdata);
        case 'z'
            i = i+1;
            to_in = uint32(str2num(str2mat(varargin(i))));
            procdata(:,1:to_in) = zeros(m,to_in);
        case 's'
            procdata = crosstalkrem(procdata);
        case 'c'
            i = i+1;
            w = str2num(str2mat(varargin(i)));
            procdata = cluttermap(w,procdata);
        case 'f'
            procdata = fft_gaussian(procdata);
        case 't'
            i = i+1;
            threshold = str2num(str2mat(varargin(i)));
        case 'd'
            dist = 1;
        case 'q'
            matr = 1;
        case 'm'
            i = i+1;
            multi = 1;
            peaksep = uint32(uint32(str2num(str2mat(varargin(i))))/3.9);
            peaks = zeros(m,10);
        otherwise
            disp 'Invalid argument'
    end;
    i = i+1;
end

```

```
59 if threshold ≠ 0
60     threshold = max(max(procdata))*threshold/100;
61 end
62
63
64 for i=1:m,
65     [amp peak] = findpeaks(abs(procdata(i,:)),'sortstr','descend',...
66                     'npeaks',10,'minpeakdistance',peaksep);
67     if multi == 0
68         if amp(1) > threshold
69             peaks(i) = peak(1);
70         end
71     else
72         for j=1:numel(amp)
73             if amp(j) > threshold
74                 peaks(i,j) = peak(j);
75             else
76                 break;
77             end
78         end
79     end
80 end
81
82
83 if dist == 1
84     peaks = peaks*3.9;
85 end
86
87 if matr == 1
88     if multi == 0
89         peaks = reshape(peaks,16,uint32(m/16))';
90     else
91         tmp = zeros(m/16,16,10);
92         l = 1;
93         for i=1:m/16
94             for j=1:16
95                 tmp(i,j,:) = peaks(l,:);
96                 l = l+1;
97             end
98         end
99         peaks = tmp;
100    end
101 end
102
103 end
```

Listing 2: crosstalkrem.m

```
1 function [ outdata ] = crosstalkrem( indata )
2 % Clutter reduction by using SVD, can be called with entire dataset or as
3 % partial
4
```

```
[u,s,v] = svd(indata);
s(1,1) = 0;
outdata = u*s*v';

end
```

Listing 3: cluttermap.m

```
function [ outdata ] = cluttermap( weight,data )
% An adaptive cluttermap

[m,n] = size(data);
cluttermap = zeros(1,n);
outdata = zeros(m,n);

for i=1:m
    for j=1:n,
        cluttermap(j) = (1-weight)*cluttermap(j)+data(i,j)*weight;
        outdata(i,j) = data(i,j)-cluttermap(j);
    end
end

end
```

Listing 4: rangecompensation.m

```
function [ outdata ] = rangecompensation( indata )
% Range compensation factor, function: y = 1 + 0.007i ? 0.000003i^2

[m,n] = size(indata);
outdata = zeros(m,n);

for j=1:m
    for i=1:n,
        if (i < n/2)
            outdata(j,i) = indata(j,i) * (1+(0.006*i));
        else
            outdata(j,i) = indata(j,i) * (4.07+(0.002*i));
    end
end

end
```

Listing 5: fft_gaussian.m

```
function [ outdata ] = fft_gaussian( data )
% Frequency-domain bandpass filter using bell curve

[m,n] = size(data);
```

```
outdata = zeros(m,n);

for i=1:m
    outdata(i,:) = fft(data(i,:));

    for j=1:n
        outdata(i,j) = outdata(i,j)*gaussian(j,180,0.3); %150,0.3
    end
    outdata(i,:) = abs(ifft(outdata(i,:)));
end


function val = gaussian(n,N,sigma)
    val=exp(-0.5*2^(( (n-(N-1)/2.0) / (sigma*(N-1)/2.0)) ));
end

end
```

Listing 6: sar.m

```
function [ map ] = sar( resolution, mapx, mapy, robot_position, ...
    radar_data )
% Back projection algorithm
%   resolution = gridsize in cm
%   mapx = [xmin xmax] in m
%   mapy = [ymin ymax] in m
%   robot_position = [x y alpha] positions of agent, in m and radians
%   radar_data = data to SAR process

xdim = uint32( (mapx(2)-mapx(1))*100/resolution );
ydim = uint32( (mapy(2)-mapy(1))*100/resolution );

map = zeros(xdim, ydim);

xindex2meters = linspace(mapx(1),mapx(2),xdim);
yindex2meters = linspace(mapy(1),mapy(2),ydim);

robotdim = length(robot_position);

progressbar(0,0);

for i=1:xdim
    progressbar(double(double(i)/double(xdim)),[]);
    drawnow;
    for j=1:ydim
        progressbar([],double(double(j)/double(ydim)));
        drawnow;
        dataindex = 1;
        for k=1:robotdim
            % check if robot ≤ 4m from point
            if distancePoints( [xindex2meters(i) yindex2meters(j)], ...
                    robot_position(k,1:2) ) ≤ 4
```

```
                a = -90;
                for l=1:16
                    % check if point is in field-of-view of agent
                    if isinsector(robot_position(k,1:2), ...
                            robot_position(k,3)+deg2rad(a), ...
                            [xindex2meters(i) yindex2meters(j)])
                        map = add2map(radar_data(dataindex,:),[i j], ...
                            robot_position(k,1:2), map);
                    end
                    dataindex = dataindex + 1;
                    a = a+11.25;
                end
            else
                dataindex = dataindex + 16;
            end
        end
    end
end


    function ret = isinsector(pos, alpha, point)
        %find cooordinates in sector, return true if point is contained
        p1 = pos;
        p2 = [(4*cos(alpha-(deg2rad(20))) + pos(1)) ...
            (4*sin(alpha-(deg2rad(20))) + pos(2))];
        p3 = [(4*cos(alpha+(deg2rad(20))) + pos(1)) ...
            (4*sin(alpha+(deg2rad(20))) + pos(2))];
        ret = isPointInTriangle(point, p1, p2, p3);
    end

    function map = add2map(data, point, robotpos, map)
        %add the return from point in data to map
        d = distancePoints( [robotpos(1) robotpos(2)], ...
            [xindex2meters(point(1)) yindex2meters(point(2))] );
        map(point(1),point(2)) = map(point(1), ...
            point(2)) + abs(data(uint32( (d*1000)/3.906) ));
    end
end
```

Listing 7: gui.m

```
function [  ] = gui( ip )
% GUI for communicating with the SLAMbot

robot = connect(ip,25,2);
rw = robot.getOutputStream;


figure('Position',[500 1 450 300]);


% control buttons
```

```
startslam_but = uicontrol('Style', 'pushbutton', 'String', 'start',...
    'Position', [20 265 150 20],...
    'callback',@(varargin) startslam,...
    'userdata',0);
stopslam_but = uicontrol('Style', 'pushbutton', 'String', 'stop',...
    'Position', [20 240 150 20],...
    'callback',@(varargin) stopslam,...
    'userdata',0);
speed_text = uicontrol('Style', 'text', 'String', 'speed',...
    'Position', [20 205 150 15],...
    'callback','set(gcbo,''userdata'',1)',...
    'userdata',0);
speed_edit = uicontrol('Style', 'edit', 'String', '105',...
    'Position', [20 180 150 25],...
    'callback',@(varargin) setspeed,...
    'userdata',0);
data_but = uicontrol('Style', 'pushbutton', 'String', 'get sensordata',...
    'Position', [20 135 150 20],...
    'callback',@(varargin) getdata,...
    'userdata',0);
exit_but = uicontrol('Style', 'pushbutton', 'String', 'exit',...
    'Position', [20 110 150 20],...
    'callback','set(gcbo,''userdata'',1)',...
    'userdata',0);

% direction buttons
forward_but = uicontrol('Style', 'pushbutton', 'String', '^',...
    'Position', [310 200 50 50],...
    'callback',@(varargin) drive('f#0#'),...
    'userdata',0);
backward_but = uicontrol('Style', 'pushbutton', 'String', 'v',...
    'Position', [310 80 50 50],...
    'callback',@(varargin) drive('b#0#'),...
    'userdata',0);
right_but = uicontrol('Style', 'pushbutton', 'String', '>',...
    'Position', [370 140 50 50],...
    'callback',@(varargin) drive('r#0#'),...
    'userdata',0);
left_but = uicontrol('Style', 'pushbutton', 'String', '<',...
    'Position', [250 140 50 50],...
    'callback',@(varargin) drive('l#0#'),...
    'userdata',0);
stop_but = uicontrol('Style', 'pushbutton', 'String', 's',...
    'Position', [310 140 50 50],...
    'callback',@(varargin) drive('s#'),...
    'userdata',0);

setspeed;


while get(exit_but,'userdata') == 0
    %main loop: not much here, as the callbacks runs the show
    waitforbuttonpress;
```

```
65      pause(0.5);
66      drawnow;
67  end
68
69      function setspeed()
70          speed = get(speed_edit,'string');
71      end
72
73      function drive(dir)
74          ssend(rw,strcat(dir,speed));
75      end
76
77      function startslam()
78          ssend(rw,strcat('slam-start#0#',speed));
79      end
80
81      function stopslam()
82          ssend(rw,'slam-stop');
83      end
84
85      function getdata()
86          %obsolete
87      end
88
89  end
```

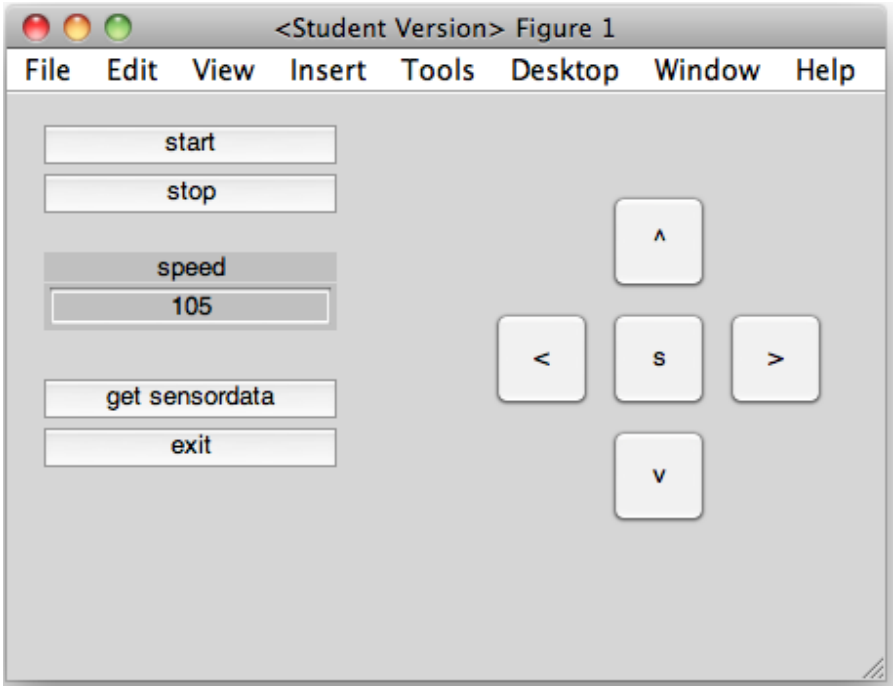


Figure 1: Screenshot of the GUI used to communicate with the agent.

Appendix B:
SLAMbot schematic

The full schematic of the SLAMbot connection board. The PCB was used for connecting servos, stepper motor and rotary encoders to the Beaglebone micro computer, as well as distribute power from a battery pack to the robot and sensors.

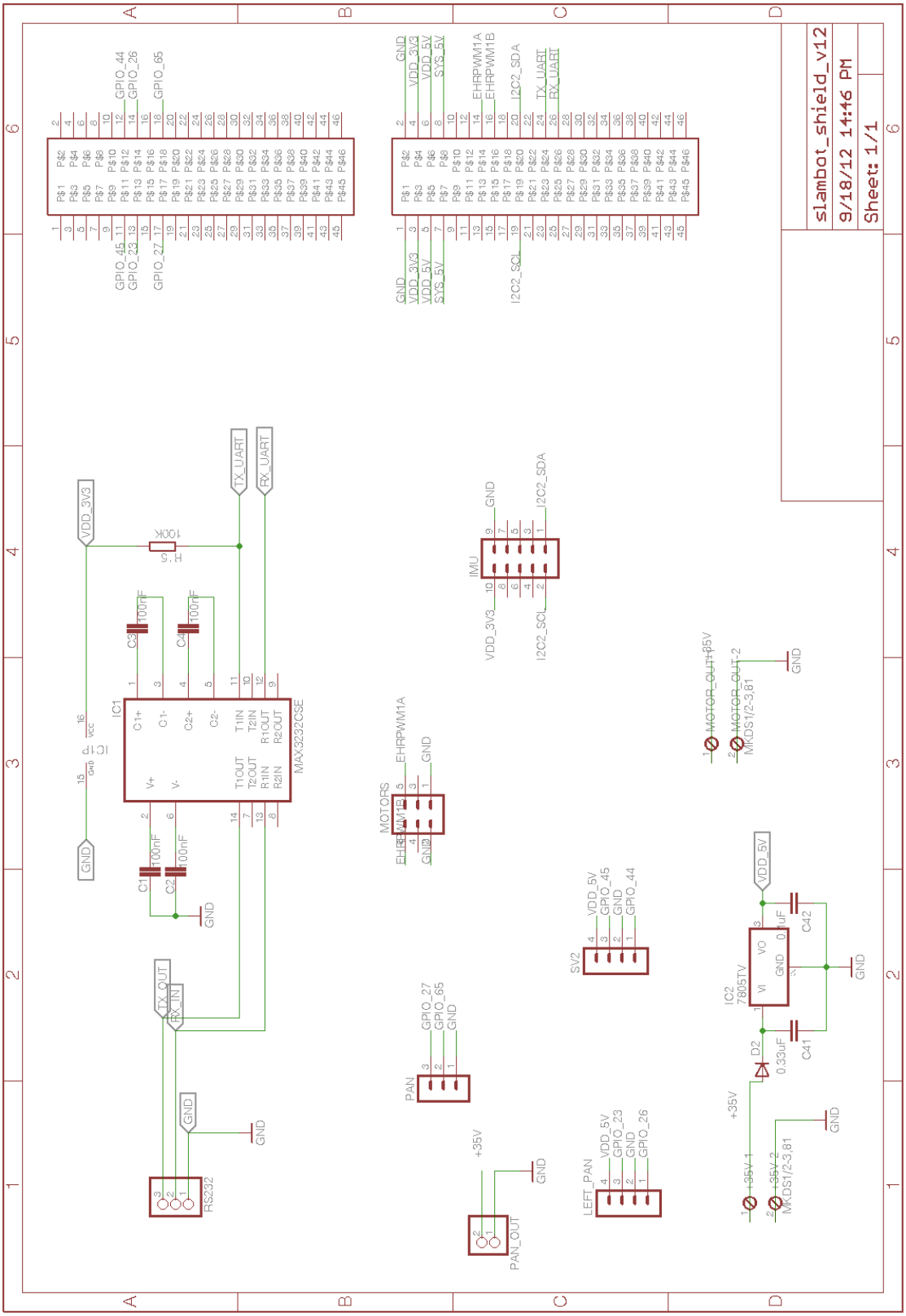


Figure 2: Schematic of the SLAMbot connection board.

List of Figures

List of Tables

List of Acronyms

EKF	Extended Kalman Filter
EM	Electromagnetic
EMW	Electromagnetic Waves
FT	Fourier Transform
GPIO	General Purpose Input/Output
IMU	Inertial Measurement Unit
kNN	k Nearest Neighbors
MLP	Multi-Layer Perceptron
PCB	Printed Circuit Board
RADAR	RAdio Detection and Ranging
RF	Radio Frequency
SAR	Synthetic Aperture Radar
SLAM	Simultaneous Localization and Mapping
SNR	Signal-to-Noise Ratio
SVD	Singular Value Decomposition
SVM	Support Vector Machine
UWB	Ultra Wide Band
UWBIR	Ultra Wide Band Impulse Radar

Bibliography

[1] M. Aftanas. *Through Wall Imaging with UWB Radar System*. PhD thesis, Technical University of Kousice, 2009.

[2] K. O. Arras. CAS-Toolbox. http://www.cas.kth.se/toolbox/, 2003. Visited: 27.08.2012.

[3] K. O. Arras. *Feature-Based Robot Navigation in Known and Unknown Enviroments*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2003.

[4] L. Y. Astanin and A. A. Kostylev. *Ultrawideband Radar Measurements*. The Institution of Electrical Engineers, 1997.

[5] T. Bailey and H. Durrant-Whyte. Simultaneous Localization and Mapping: part I. *Robotics Automation Magazine, IEEE*, 13(2):99 –110, june 2006.

[6] T. Bailey and H. Durrant-Whyte. Simultaneous Localization and Mapping (SLAM): part II. *Robotics Automation Magazine, IEEE*, 13(3):108 –117, sept. 2006.

[7] D. Brescianini. Ultra Wideband Radar for Micro Aerial Vehicles. Master's thesis, Swiss Federal Institute of Technology Zürich, 2009.

[8] J. L. Devore and K. N. Berk. *Modern Mathematical Statistics with Applications*. Thomson Brooks/Cole., 2007.

[9] A. Eliazar and R. Parr. DP-SLAM. http://www.cs.duke.edu/~parr/dpslam/, 2003. Visited: 16.10.2012.

[10] P. Elinas, R. Sim, and J.J. Little. σSLAM: Stereo Vision SLAM using the Rao-Blackwellised Particle Filter and a Novel Mixture Proposal Distribution. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1564–1570, 2006.

[11] R.J. Fontana. Recent System Applications of Short-Pulse Ultra-Wideband (uwb) Technology. *Microwave Theory and Techniques, IEEE Transactions on*, 52(9):2087 – 2104, sept. 2004.

[12] G. Franceschetti and R. Lanari. *Synthetic Aperture Radar Processing*. CRC Press, 1999.

[13] A. Gelb. *Applied Optimal Estimation*. MIT Press, 1974.

[14] Guizzo, E. and Deyle, T. Robotic Trends for 2012. http://spectrum.ieee.org/automaton/robotics/robotics-hardware/robotics-trends-for-2012, 2012. Visited: 27.08.2012.

[15] Texas Instruments. BeagleBone. http://beagleboard.org/bone, 2012. Visited: 01.11.2012.

[16] Intelligent Agent. D1S Radar Sensor. http://www.intelligentagent.no/wp-content/uploads/2012/05/D1S_ProductSheet_v0.6.pdf, 2012. Visited: 27.08.2012.

[17] D. H. Johnson and D. E. Dudgeon. *Array Signal Processing: Concepts and Techniques*. Prentice-Hall, Inc., 1993.

[18] M. Kronauge and H. Rohling. Clutter map for a CW surveillance radar. In *Radar Symposium (IRS), 2012 13th International*, pages 133 –136, may 2012.

[19] Yangming Li and E.B. Olson. Extracting General-Purpose Features from LIDAR Data. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1388–1393, 2010.

[20] S. Marsland. *Machine Learning*. CRC Press, 2009.

[21] Rapid-I. RapidMiner data mining software. http://www.rapidminer.com, 2013. Visited: 06.02.2013.

[22] Rover. Lynxmotion 4WD Rover. http://www.lynxmotion.com/c-111-no-electronics-kit.aspx, 2012. Visited: 27.08.2012.

[23] L. Sakkila, A. Rivenq, C. Tatkeu, Y. El Hillali, J.-P. Ghys, and J.-M. Rouvaen. Methods of Target Recognition for UWB Radar. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 949 –954, june 2010.

[24] UAS Vision. UK Deploys Black Hornet UAS in Afghanistan. http://www.uasvision.com/2013/02/05/uk-deploys-black-hornet-nano-uas-in-afghanistan/, 2013. Visited: 03.04.2013.

[25] Zheng Weiqin. Sonar Features Extraction Algorithm for a Mobile Robot. In *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, volume 3, pages 689–692, 2009.

[26] B. Yamauchi. Fusing Ultra-Wideband Radar and LIDAR for Small UGV Navigation in All-Weather Conditions. Technical report, iRobot Corp., 2010. http://www.robotfrontier.com/papers/daredevil-spie2010.pdf.